# Desarrollo de un entorno de virtualización de redes con fines docentes


UNIVERSIDAD
COMPLUTENSE
MADRID

Víctor Fernández Duque

Director - Juan Carlos Fabero Jiménez

Trabajo de fin de grado del Grado en Ingeniería del Software

# Resumen

El objetivo de nuestro proyecto es la creación de un controlador para usar de forma sencilla topologías virtuales de red y el estudio de las redes definidas por *software*. Para ello estudiamos el protocolo OpenFlow y su implementación más conocida Open vSwitch.

Para el desarrollo de nuestra aplicación elegimos el lenguaje Tcl-Tk debido a la facilidad para la creación de la interfaz gráfica y la simplicidad del lenguaje mediante *script*, para ello fue necesario el estudio de sus peculiaridades.

Nuestra aplicación es capaz de generar una topología de red y la gestión del flujo de datos de estos mediante la creación, eliminación y gestión de *bridges* virtuales, interfaces físicos, *tuntap* y enlaces *vlink*. Además de tener la capacidad de generar archivos para guardar y cargar los estados de la misma.

Por estas razones pensamos que el proyecto cumple las expectativas iniciales ya que hemos entendido desde la base el funcionamiento de las redes definidas por *software* y hemos podido desarrollar nuestra aplicación para la creación de nuestros entornos virtuales.

# Palabras clave

SDN, Openflow, Open vSwitch, Tcl-Tk, *bridge*, topología de red, flujo de datos, virtualización, protocolos de red, controlador.

# Abstract

The purpose of this project is the creation of a controller to use virtual topologies of network and the investigation of the networks based and defined by software. For that matter, we study the Openflow protocol and it's most known implementation Open vSwitch.

For this application's development we use the Tcl-Tk language, due to the easiness of the creation of graphic interfaces and the simplicity of the language through the use of scripts, and for that was necessary the study of its peculiarities.

Our application is capable of generate a network topology, the management of data flow through the creation, elimination and management of virtual bridges, physical interfaces, tuntaps and Vlink links. Furthermore, having the capacity of generating files in order to save and load the states of itself.

Because of that reasons, we consider the project meets the main expectations due the fact that we understood since the beginning the performance of the networks defined by software and we have been able to develop our application for the creation of our virtual environments.

# Keywords

SDN, Openflow, Open vSwitch, Tcl-Tk, bridge, network topology, data flow, virtualization, network protocols, controller.

# 1. Introducción

Nuestro objetivo es la investigación de las redes definidas por *software* [1][2], para estudiar la creación de redes y máquinas virtualizadas. Para ello nos centramos en el protocolo Openflow [2] y en su implementación más popular Open vSwitch [2], la cual nos permite la creación y gestión de *bridges* virtuales y flujos de datos.

La razón por la que se decidió hacer esta investigación es debido a que el alumno no había recibido ninguna asignatura durante la carrera sobre este rama de conocimiento, y además, inconscientemente, llevaba poniéndola en práctica durante los cuatro años de la misma, mediante un servidor que tenía contratado en Francia donde implementó todas las aplicaciones que tuvo que desarrollar. En éste, para evitar sobrecostes debido al poco presupuesto, se usaron sistemas de virtualización de máquinas y redes para, de esta forma, poder compartir costes con otras personas.

Los sistemas que se usaron fueron VMware [3] y Proxmox [4], los cuales eran de pago pero permitían pruebas para máquinas pequeñas. Este fue el principio debido a que aunque la rama de estudio del alumno estaba más ligada al *software* que a redes despertó curiosidad en él para saber en qué se basan estos complejos sistemas de pago y cómo podría usarlos él mismo empezando desde la base.

Por lo tanto buscamos a un profesor el cual permitiera realizar este estudio y que aportase ayuda en toda la investigación. Por eso nos pusimos en contacto con el profesor Juan Carlos Fabero Jiménez, el encargado de guiarnos durante todo este año.

Hemos conseguido entender la base de las redes definidas por *software*, estudiar los inicios del protocolo Openflow el cual quiere llegar a convertirse en un estándar para todos los fabricantes, ya que éste no se limita solo al ámbito académico sino que se aplica también al profesional. Además hemos aprendido el funcionamiento de Open vSwitch y todas las posibilidades que éste nos ofrece.

Como agregado, decidimos no programar nuestra aplicación en un lenguaje común de la actualidad pudiendo ser Java, Python o C++ sino que estudiamos cómo implementarla mediante Tcl-Tk [5], debido a su portabilidad, como explicaremos más adelante.

Nuestro objetivo queda cumplido ya que hemos conseguido sintetizar todo lo aprendido creando nuestra aplicación, la cual facilita el uso de Open vSwitch mediante una interfaz gráfica sencilla.

# 2. Requisitos

Desde el principio del proyecto han existido dificultades notables. Los requisitos principales fueron entender bien qué es una SDN (*Software Defined Networks*, Redes definidas por *software*). Después de entender esto descubrimos OpenFlow y Open vSwitch. Tras estudiar estos dos conceptos investigamos sobre Tcl-Tk para desarrollar nuestro controlador. El problema de lo anterior es que no ha sido un objetivo de la rama informática que ha cursado el alumno por lo tanto se ha necesitado bastante tiempo de estudio para comprender todos los conceptos, desde hacer en papel esquemas de topologías de red, a estudiar cómo funciona el lenguaje Tcl-Tk.

## 2.1 Redes definidas por *software*

La primera pregunta que nos hacemos es qué son las SDN. Como hemos dicho con anterioridad se denominan *redes definidas por software*, mediante las cuales se separa la capa de control de la capa de encaminamiento. Permiten separar la topología lógica de la infraestructura física. Las aplicaciones de usuario se comunican mediante la API (*Application Programming Interface*, interfaz de programación de aplicaciones) con la capa de control y ésta mediante el protocolo OpenFlow, define la infraestructura o capa de datos. Aunque está destinada a emplearse en entornos virtuales también es compatible con entornos físicos. Este sistema puede ser una buena oportunidad para un entorno educativo debido a los escasos recursos que son necesarios.

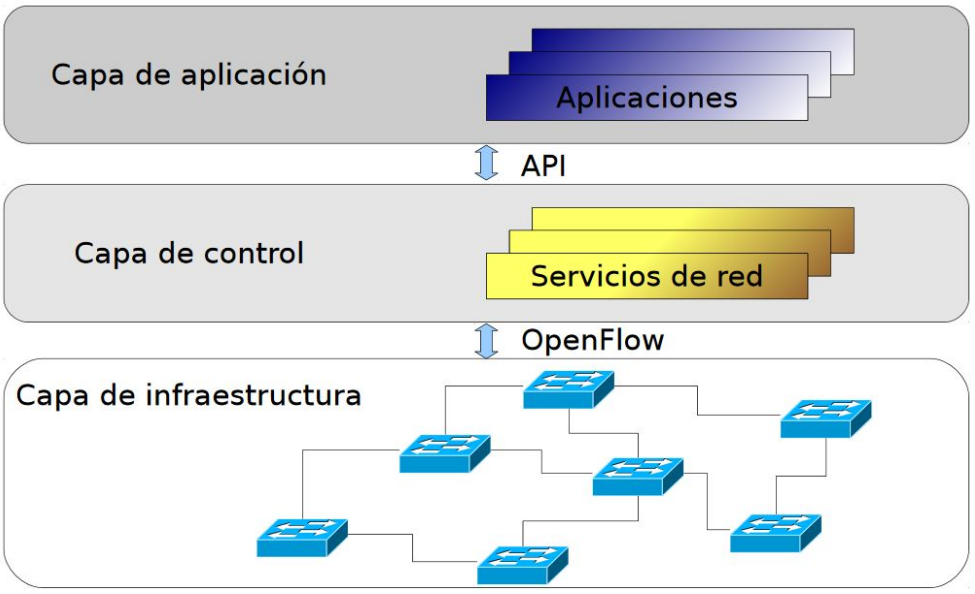


Figura 2.1 Arquitectura SDN

En la figura 2.1 mostramos lo definido anteriormente, separamos la capa de aplicación de la de control, nos comunicamos mediante una API y con OpenFlow creamos toda la estructura de red o capa de datos.

Un punto importante por destacar, como comentamos con anterioridad, es que las redes SDN fueron destinadas principalmente a un entorno virtualizado, pero la gran ventaja que ofrecen a día de hoy es que muchos fabricantes lo han llevado a su hardware por lo que se amplía bastante el ámbito de aplicación de este paradigma.

## 2.2 Redes VLAN

En este apartado hablaremos de las redes VLAN (*Virtual Local Area Network*, Red de área local virtual), las cuales clasifican las máquinas por grupos. Cuando un usuario manda un paquete a la red, el administrador configura los puertos para asignar a cada usuario una VLAN, es decir, solo podrán comunicarse las máquinas que se encuentran en la misma.

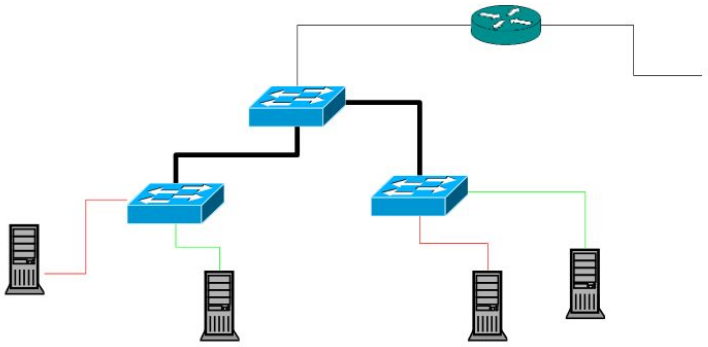

Figura 2.2 Red VLAN

Por ello, tendríamos un control de red mayor, ya que aunque tuviéramos varias máquinas en una misma red física, podremos elegir si estas pueden comunicarse entre sí. En la figura 2.2, tenemos cuatro máquinas conectadas a dos VLAN, una roja y otra verde. Estas máquinas sólo podrán comunicarse con la que se encuentre en su misma VLAN.

Todo esto se hace posible mediante el protocolo *802.1Q*. Las máquinas conectadas a una red no tienen conocimiento de lo que ocurre. La red es controlada por un administrador, que es quien asigna el grupo al puerto donde se conecta el usuario definiendo, de esta forma, la VLAN. Cuando enviamos un paquete de datos, este, al llegar al primer conmutador modifica la trama agregando la etiqueta de VLAN.

A continuación, mostraremos el funcionamiento del protocolo *802.1Q:*


Figura 2.3 Trama de datos de usuario.


Figura 2.4 Trama de datos con etiqueta 802.1Q.

En la figura 2.3 vemos la trama de datos que envía el usuario por la red.

En la figura 2.4 podemos observar la trama del usuario además de la etiqueta *802.1Q* que añade el *bridge*. Esta etiqueta está compuesta por 32 bits, los cuales son:

- 16 bits para el campo de tipo, 0x8100.
- 4 bits para la prioridad.
- 12 bits para el identificador de la VLAN, permite 4096 identificadores distintos (0...4095).

Las etiquetas no cambian a lo largo de todo el recorrido. El último conmutador elimina la etiqueta y los dispositivos pueden recibir la trama inicial.

VLAN está limitada, en otras palabras, permite un número máximo de agrupaciones y no tiene control de los datos que pasan por la red. *OpenFlow* soluciona los problemas anteriores. Quiere convertirse en el protocolo estándar en la definición de redes mediante *software*, permitiendo la gestión de *bridges* de forma

remota. Nosotros estudiamos *Open vSwitch* que es una de las implementaciones más populares de OpenFlow para mostrar cómo funciona esta tecnología enseñaremos lo que tuvimos que estudiar.

## 2.3 Open vSwitch

*Open vSwitch* es un *software* que implementa un *bridge* virtual en entornos de servidores virtualizados. Permite la comunicación de máquinas virtuales que se ejecutan en un mismo entorno físico, controlan el acceso de las mismas a la red física, además de permitir el manejo de interfaces físicas con la misma arquitectura. Un ejemplo de todo esto lo podemos ver en la figura 2.5.

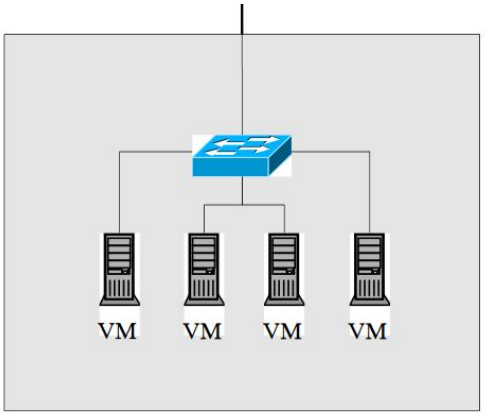


Figura 2.5 Bridge virtual.

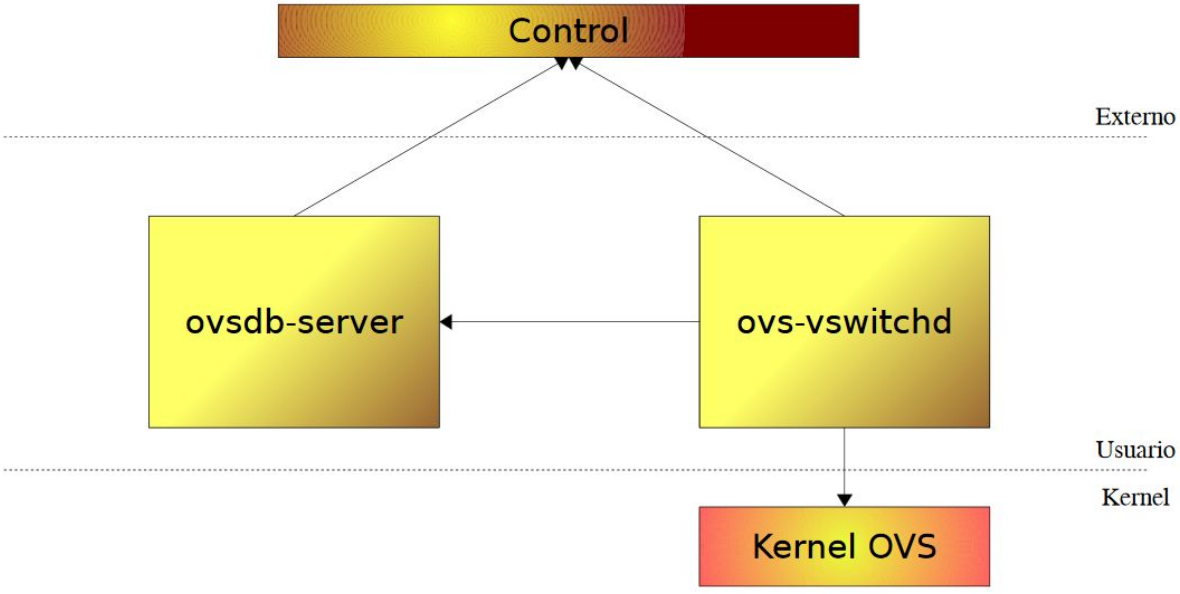


Figura 2.6 Arquitectura open vSwitch.

En la figura 2.6 mostramos la arquitectura de Open vSwitch. Éste posee tres partes distinguibles la primera es el controlador con el cual interactuamos con la aplicación. La segunda es la capa donde se sitúa la base de datos *ovsdb-server* y el gestor del programa *ovs-vswitchd*. Por último el módulo del kernel, llamado Kernel OVS.

## 2.4 El gestor de la base de datos ovsdb-server

La base de datos de la aplicación se encuentra en *ovsdb-server*, donde ésta almacena toda la información como puentes, interfaces, puertos y túneles con los cuales crearemos nuestra topología. Su función es conservar toda la información de manera no volátil. Podemos ver la estructura de la misma en la figura 2.7.

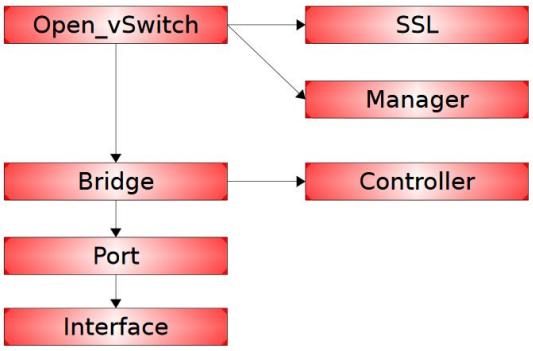


Figura 2.7 Estructura de ovsdb-server.

El gestor de la aplicación *ovs-vswitchd* tiene distintas funciones:

- Mantener actualizada la información de la base de datos.
- Configurar el demonio que controla el módulo del kernel.
- Gestionar las interfaces virtuales.
- Control del flujo de datos entre *bridges*.

Además posee algunas características especiales:

- Soporta múltiples *bridges* virtuales.
- Permite *mirroring*, *trunking*, *vlan* y *patching*.


Figura 2.8 Distintas formas de utilización de Open vSwitch.

En la figura 2.8 mostramos un pequeño ejemplo de lo que se puede conseguir con Open vSwitch. Aquí se muestran distintos puertos conectados a dos *bridges*, los cuales están configurados de distintas formas. Un ejemplo es *mirror*, por donde refleja todo el tráfico que pasa por el *bridge* o pr1, el cual está en modo *trunking*, con

el que conseguimos el doble de ancho de banda y además tiene distintas VLAN diferenciadas por color.

En la figura 2.9, en la parte izquierda, el módulo del kernel actúa de forma habitual como un *bridge*. Por otra parte, en la parte derecha, observamos cómo *Open vSwitch* utiliza el gestor para filtrar los paquetes. Esto supone un sobrecoste, por lo que el gestor *ovs-vswitchd* solo toma la primera trama de datos y configura el módulo del kernel para que sepa cómo tratar ese tipo de paquetes. Esto se realiza hasta que expire el tratamiento del mismo o se produzca un cambio en la configuración para tratar el flujo de datos.

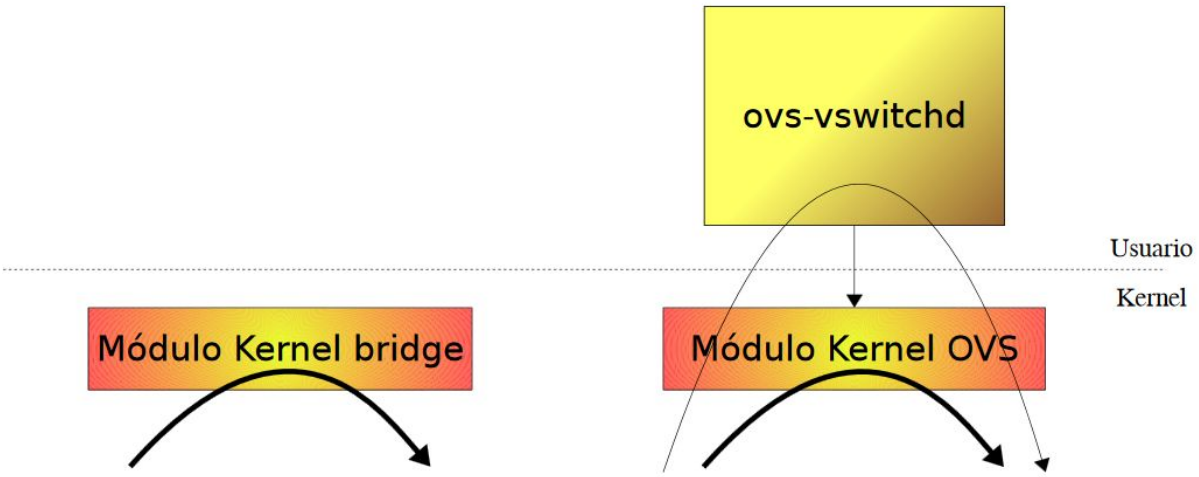


Figura 2.9 Funcionamiento del gestor y el demonio.

Expuesto el funcionamiento, definiremos el alcance del sistema explicando de esta forma las funciones disponibles.

El controlador nos ofrece dos comandos principales, con los cuales podemos gestionar la creación de la topología de red y el control de flujos de la misma: *ovs-vsctl* y *ovs-ofctl,* los cuales explicaremos a continuación.

## 2.5 El gestor de la aplicación ovs-vswitchd

Para configurar el gestor de Open vSwitch (*ovs-vswitchd),* utilizamos el comando ovs-vsctl. Éste nos ofrece, entre otras, las siguientes funciones:

- Crear un puente (*bridge*).
- Añadir un puerto al puente.
- Listar los puentes.
- Listar los puertos de un puente.
- Listar información de un puerto.

- Mostrar información de un interfaz.
- Listar información de la base de datos.

Mediante *patching* podemos unir varios *bridges* con el fin crear topologías complejas. En *GNU/Linux* podemos crear enlaces *Ethernet* virtuales mediante la utilidad *iproute2.* Con esto además conseguiremos analizar el tráfico que pasa por estos enlaces mediante otra aplicaciones, como por ejemplo *Wireshark*.

Cuando se crean topologías complejas existe la posibilidad de crear bucles y, por tanto, producir bloqueos en nuestra red. Para evitar esta saturación se usa *STP* (Spanning Tree Protocol), con el cual se crea un árbol sin bucles entre los *bridges,* dejando algunos puertos bloqueados y por los cuales solo pasará tráfico de sincronización, hasta que sea necesario utilizarlo porque haya cualquier otra necesidad o una rama se encuentre desconectada o se haya producido algún otro cambio en la topología.

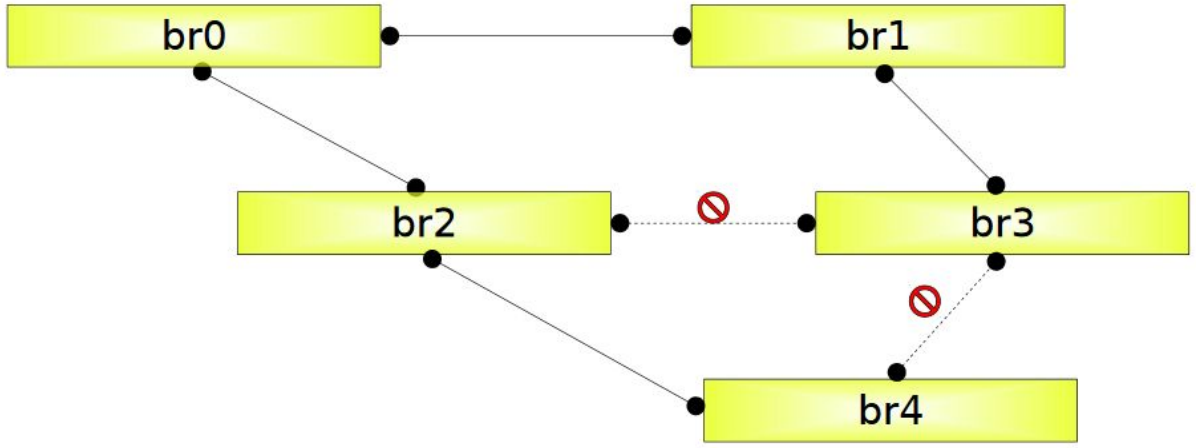


Figura 2.10 Bridges virtuales con protocolo STP.

En la figura 2.10 mostramos un ejemplo de topología compleja con protocolo STP, en la cual se muestra que algunos puertos quedan sin usar hasta que éstos sean necesarios, evitando de esta forma bucles.

Open vSwitch permite también el protocolo 802.1Q. En la siguiente figura se muestra un ejemplo:

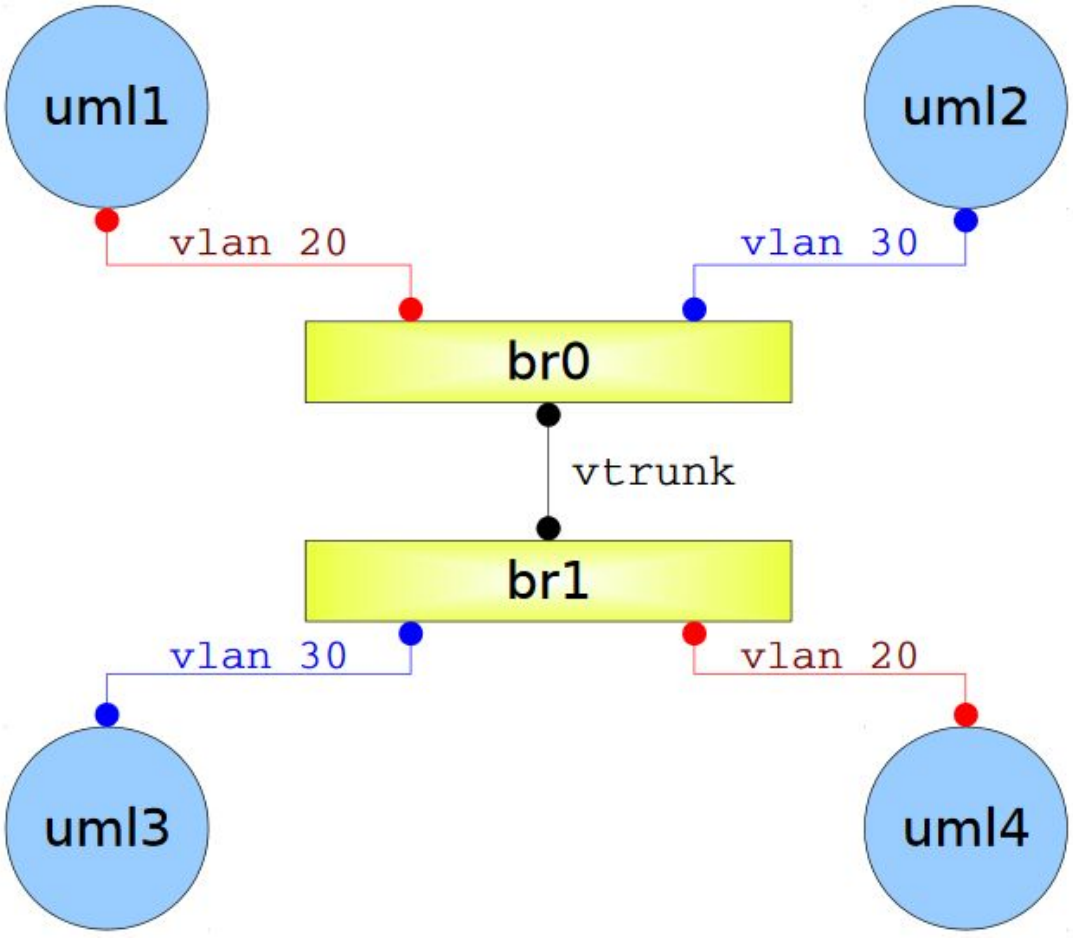


Figura 2.11 Protocolo 802.1Q en Open vSwitch.

En la figura 2.11 se muestra como Open vSwitch gestiona el protocolo 802.1Q, en la que tenemos dos grupos distintos: el 20 y 30, donde las máquinas virtuales están conectadas a dos *bridges* vinculados mediante un cable de *Ethernet* virtual. Estas máquinas solo se puede comunicar con su homóloga de cada *bridge,* la cual está en su misma VLAN.

El potencial de *Open vSwitch* reside en la posibilidad de poder controlar de forma dinámica los flujos de datos que atraviesan un *bridge*. Un flujo se puede entender como una regla específica de cómo deben tratarse los paquetes que cumplen determinados criterios.

Las distintas reglas se agrupan en tablas que van desde 0 a 253, y se ordenan por prioridad, siendo la de menos peso la más prioritaria.

Cada regla tiene una sección “*match”* y una sección “*actions*”:

- *Match* se refiere a la condición que busca. Puede afectar desde un protocolo a una interfaz. Por ejemplo: `in_port`, `dl_src`, `dl_dst`, `dl_vlan`, `dl_type`, `ip`, `arp`, `icmp`, `tcp`, `udp`, `ipv6`, `icmp6`, `tcp6`, `udp6`.

- *Actions* se refiere a qué debe hacer cuando encuentre esa condición específica. Por ejemplo: `normal`, `flood`, `all`, `drop`, `resubmit`, `mod_vlan_vid`, `strip_vlan`, `mod_dl_src`, `mod_dl_dst`, `goto_table`.

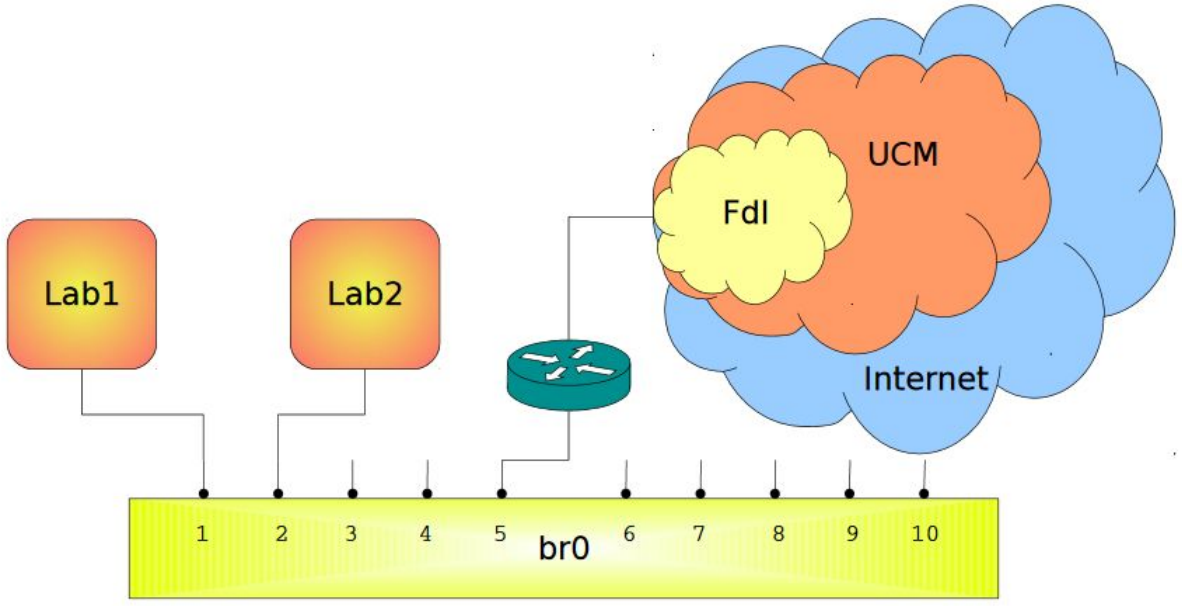


Figura 2.12 Posible configuración de red de la facultad.

Como se muestra en la figura 2.12, exponemos un prototipo de la configuración de la red de la Facultad. Mostramos cómo tendríamos conectados los laboratorios de nuestra Facultad a un *bridge* y éstos, a su vez, estarían conectados a la red de la Facultad, de la Universidad y a Internet. Un caso específico podría ser el de poner los laboratorios en modo examen, bloqueando el acceso a Internet de los computadores de los alumnos. Como todos los ordenadores están dentro de la tabla del laboratorio, sería un proceso tan sencillo como el de bloquear el acceso a esa tabla para restringir el acceso a Internet.

Por lo tanto, las tablas permiten organizar de manera eficiente los flujos. Una configuración posible sería la siguiente:

- Tablas 0: clasificador.
- Tablas 100+x donde x es el número del laboratorio. Cada uno de los puertos del *bridge*.

- Tabla 200 para los servidores de la Facultad.
- Tabla 210 para los servidores de la Universidad.
- Tabla 220 para la salida a Internet.

Queda demostrado que este sistema nos da miles de combinaciones posibles para crear nuestra red y controlar todo lo que se mueve por ella, de una forma simple y eficiente.

## 2.6 Reglas QoS

Es posible limitar el ancho de banda mínimo y máximo asignado a cada puerto, así como la prioridad y el tamaño de ráfaga entre otros. Esto se hace mediante la definición de colas mediante *ovs-vsctl* usando reglas QoS [6].

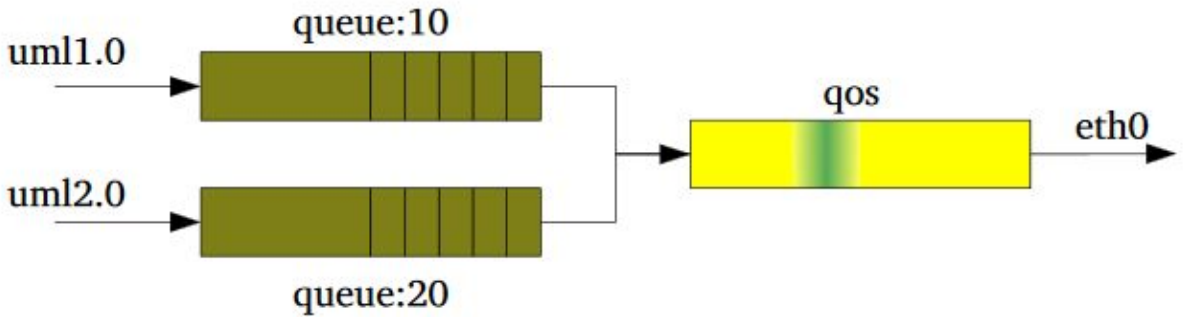


Figura 2.14 Reglas QoS.

En la figura 2.14 se muestra una interfaz física la cual tiene 1 Gbps de ancho de banda, y a ésta se conectan dos interfaces virtuales, uml1.0 y uml2.0, a los que se va a imponer una limitación de 10 Mbps y 20 Mbps respectivamente. *Open vSwitch,* mediante reglas QoS, nos permite controlar el ancho de banda de cada uno de las interfaces.

Además, también nos permite usar HTB, el cual es un algoritmo cuyas siglas hacen referencia a *Hierarchical Token Bucket*. El concepto es muy sencillo: tenemos un elemento llamado *token*, que se facilita a los usuarios conectados a un ancho de banda determinado por el administrador. Si estos consumen ancho de banda a la vez que lo reciben, no podrán jamás superar el límite suministrado por el administrador de la red. Sin embargo, si el usuario no consume a la misma velocidad que se le otorgan estos *token*, los reservará y en un momento de alta demanda de red podrá superar la velocidad límite asignada por el administrador hasta que se le acaben (tamaño de ráfaga).

Este algoritmo permite asegurar el servicio a todos los usuarios de la red según sus necesidades, así como el ancho de banda asignado. Dicho algoritmo permite, a

su vez, el uso puntual de más recursos de los asignados, siempre que en promedio no se supere.

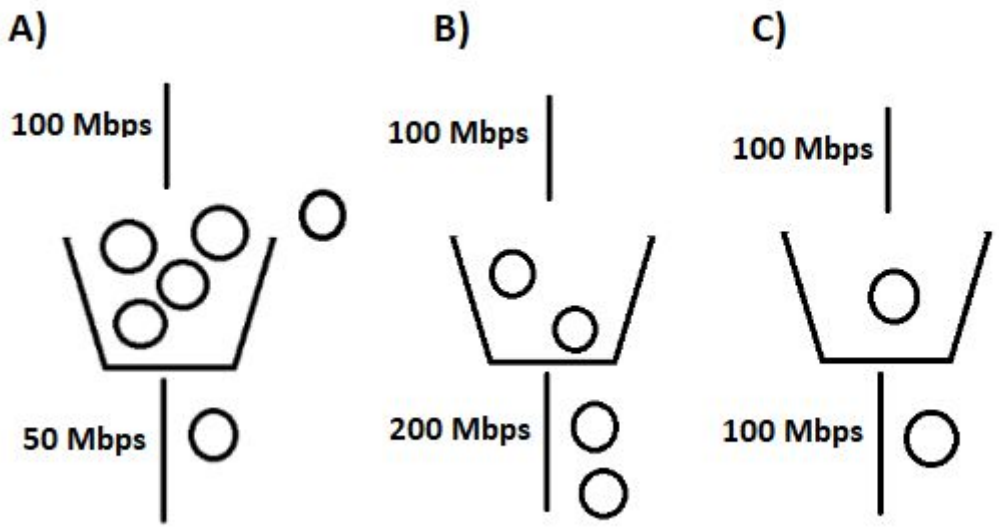


Figura 2.15 Funcionamiento HTB.

En la figura 2.15 A, el usuario recibe *tokens* a una velocidad mayor de los que él está consumiendo, por lo tanto, éste los acumula para poder usarlos en el momento que quiera. Los *token* no son ilimitados: si supera el tamaño máximo de ráfaga, estos "rebosarán" y los perderá. Esta limitación debe ser asignada por el administrador.

En la figura 2.15 B, el usuario está en una situación de mayor demanda y decide utilizar los *token* para poder hacer uso de un mayor ancho de banda y así aprovechar lo que no utilizó con anterioridad.

En la figura 2.15 C, nos encontramos en la situación en la que el usuario ha gastado todos sus *token* y vuelve al límite de ancho de banda asignado por el administrador.

## 2.7 Mirroring

El *mirroring* consiste en un proceso en el cual se copia el contenido de uno o más puertos a uno de destino, y que resulta útil para la depuración de nuestra red. A veces, es conveniente inspeccionar el tráfico que circula por alguno de los puertos, con el fin de detectar el tráfico anómalo. Para ello, podemos usar programas como *Wireshark* o algún IDS (*Intrusion Detection System*) como *snort.*

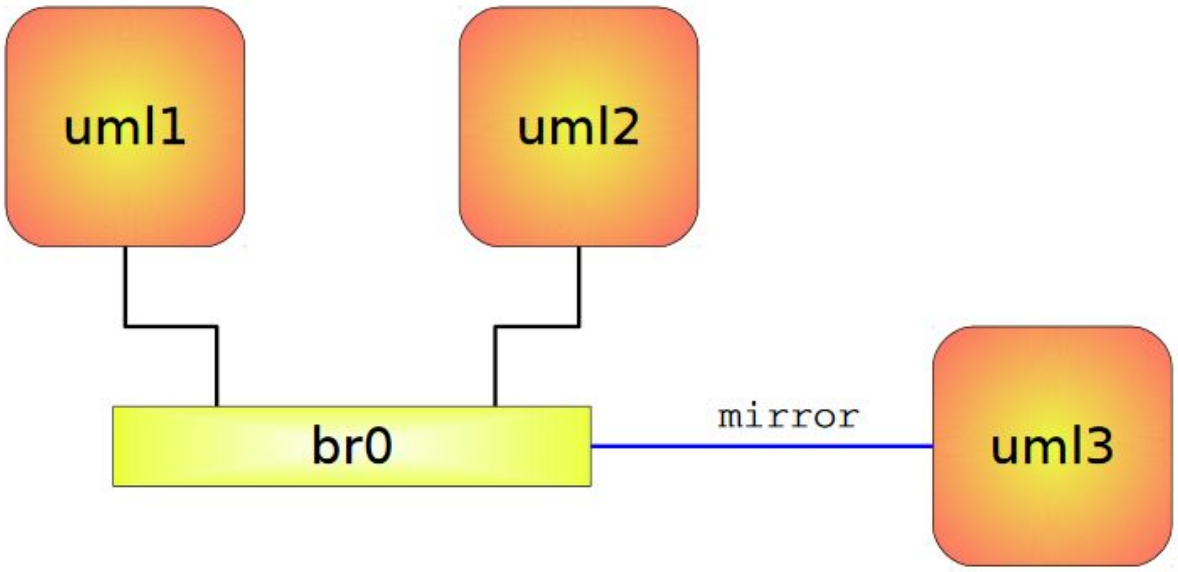


Figura 2.16 Mirroring mediante Open vSwitch.

En la figura 2.16 se puede ver cómo todo el tráfico de las máquinas uml1 y uml2 se copia al puerto de la máquina uml3. Como hemos comentado con anterioridad, éste nos puede servir para analizar las comunicaciones de nuestros usuarios.

## 2.8 Tcl-Tk

En este proyecto, decidimos hacer uso de *Tcl-Tk* [5] debido a la sencillez para crear interfaces simples y funcionales. La ventaja principal que nos ofrece es que, al ser un lenguaje interpretado, permite la prueba de código de una forma más rápida, es decir, evitando la compilación. Sin embargo, la dificultad de este lenguaje, recae principalmente en tener una semántica diferente a los lenguajes de hoy en día, como lo pueden ser *Java, Javascript* o *C++*.

# 3. Otras soluciones similares

A pesar de que en el mercado ya existen opciones para poder realizar todo lo mencionado, el objetivo de este proyecto, aunque no ofrezca innovación, es constituir un sistema propio, con el fin de estudiar el proceso desde su base y poder aprender su funcionamiento. Las alternativas principales que existen, entre otras, serían *Neutron* del proyecto *Openstack, Proxmox, VMware*, etc.

A continuación hablaremos de cada uno de los tres ejemplos expuestos.

## 3.1 OpenStack


**OpenStack** [7][11] es un sistema para la gestión de servicios en la nube el cual tiene la posibilidad de controlar grandes grupos de recursos informáticos, almacenamiento y de red en un CPD (Centro de proceso de datos). Todo está administrado desde un panel web, el cual brinda a los administradores un control absoluto y permite a los usuarios gestionar sus servicios.

Se trata de *software* libre y de código abierto, pero ésto no quiere decir que su uso se limite a entornos de prueba. OpenStack tiene marcas muy grandes que confían en el proyecto, tales como NASA (fundadores junto a *Rackspace*), *IBM, Oracle, Canonical, Cisco,* etc. Con ésto volvemos a afirmar que las SDN son, a día de hoy, una opción muy interesante de estudio, gracias a las empresas que sostienen esta tecnología. Por lo tanto, *OpenStack* posee un ecosistema fuerte, en el cual los usuarios que busquen un soporte comercial pueden elegir entre diferentes productos y servicios impulsados en el *Marketplace* (mercado de la comunidad).

La parte de red o *networking* de *OpenStack* se denomina Neutron [8], anteriormente llamado *Quantum*. Se trata de una API que permite configurar y definir la conectividad de red.

Neutron maneja la creación y administración de una infraestructura de red virtual, que incluye redes, conmutadores, subredes y encaminadores para dispositivos administrados por el servicio de cómputo *OpenStack*. También se pueden usar servicios avanzados como los cortafuegos o las redes privadas virtuales (VPN).

Este servicio de red posee el denominado servidor Neutron: Una base de datos para almacenamiento persistente y complementos, los cuales proporcionan otros

servicios como la interfaz con mecanismos de red Linux nativos, dispositivos externos o controladores SDN.

*OpenStack Networking* es completamente independiente y se puede implementar en un host dedicado. Neutron está integrado por varios componentes de *OpenStack*:

- El servicio OpenStack Identity se usa para la autenticación y autorización de solicitudes API.
- El servicio de Computación de *OpenStack* se usa para conectar cada NIC virtual en la VM a una red en particular.
- *OpenStack Dashboard* es utilizado por los administradores y los usuarios inquilinos para crear y administrar servicios de red a través de una interfaz gráfica basada en web.

A continuación mostramos desglosado todo lo que abarca:

- Soporte de red básico:
    - Ethernet.
    - VLAN.
    - Subnets y ARP.
    - DHCP.
    - IP.
    - TCP/UDP/ICMP.

- Componentes de red:
    - *Bridges*.
    - *Routers*.
    - *Firewalls*.
    - Balanceadores de carga.

- Protocolos para tunel:
    - GRE (*Generic routing encapsulation,* encapsulado de encaminamiento genérico).
    - VXLAN (*Virtual extensible local area network*, Red virtual de área local extensible)

- Espacio de nombres de red:
    - Espacio de nombres de red Linux.
    - VRF (*Virtual routing and forwarding*, encaminamiento y reenvío virtual)

- Traducción de direcciones de redes:
    - SNAT.

- DNAT.
- One to one NAT.

## 3.2 VMware


**VMware** [3][12] es la plataforma líder del mercado en virtualización para construir infraestructuras *cloud*. Puede ser considerada la opción de pago en contraposición a *OpenStack,* además de que permite la virtualización de máquinas. Nos ofrece un panel sencillo de utilizar, en el cual nos permite conectarnos a nuestro servidor y gestionar todos los recursos físicos.

*VMware Esxi* es una aplicación que nos permite la gestión de máquinas virtuales y de red. Respecto a las máquinas virtuales, se puede configurar todo: disco, CPU, RAM, tarjetas de red, etc. No notaremos que estamos en un sistema virtualizado, es decir, se asemeja a un entorno físico. Con referencia a la red, la aplicación nos aporta todas las funciones mencionadas con anterioridad: creación de *bridge* e interfaces de red virtuales, puertos virtuales, VLAN, balanceo de carga, etc. Cabe destacar que también incluye un concepto muy interesante: *IP fail over*.

*IP fail over* nos ofrece la posibilidad de contratar un paquete de IP, las cuales se pueden asignar a nuestras máquinas virtuales mediante la mac de sus interfaces de red. Ésta funcionalidad resulta útil para usuarios que quieran compartir una máquina con grandes características y dividirla de manera lógica en fracciones más pequeñas para ahorrar costes.

El inconveniente que supone el uso de *VMware* es que es una plataforma de pago, por lo tanto, puede que no esté al alcance de cualquier persona. Sin embargo, y de manera no profesional, la plataforma dispone de una licencia de prueba para servidores pequeños que no superen los 64GB de RAM. Por ello, un punto intermedio entre las dos opciones anteriores sería Proxmox, el cual introducimos en el siguiente apartado.

## 3.3 Proxmox

**Proxmox** [4][13] es un *software* de virtualización de código libre y abierto que aprovecha el sistema nativo de virtualización de Linux, contenedores KVM (*Kernel-based Virtual Machine*, máquina virtual basada en el núcleo), y, con respecto a la gestión de la red, se recurre a la utilización del *kernel* de Linux, *bridge-utils.* Por otra parte, añade la funcionalidad de un *bridge*, y además puede utilizar *Open vSwitch*, que es en lo que se basa este proyecto.

Proxmox posee una interfaz web en la cual se puede gestionar toda la creación de máquinas virtuales: discos, red, CPU, RAM, etc. Todo esto se puede hacer de una forma sencilla y desde cualquier dispositivo, gracias a que se trata de una página web. Cabe destacar que posee también un sistema de suscripciones en el cual es posible recibir asistencia y ayuda como en *OpenStack*.

## 3.4 Conclusión

Se han mencionado estos tres sistemas, y en primer lugar OpenStack y Neutron, debido a su gran uso en la industria. Por otra parte, VMware y Proxmox han sido empleados por el alumno en su uso académico.

Por lo tanto, como podemos comprobar, existen alternativas a nuestro estudio, sin embargo queríamos investigar desde la base todo lo relacionado con las SDN creando además un entorno sencillo enfocado en la docencia.

# 4. Nuestra solución. Arquitectura

En el primer momento que comprendimos todos los conceptos de las SDN, OpenFlow y en especial Open vSwitch, comenzamos a desarrollar nuestra propia solución, que debe satisfacer varias condiciones:

- Ser portable.
- Mantenible.
- Fácil de usar.
- Tener una interfaz sencilla.

## 4.1 Lenguaje

Después de algunas reuniones, consolidamos la idea de desarrollar la aplicación en el lenguaje Tcl-Tk. Lo consideramos de esta forma debido a su sencillez para crear interfaces funcionales y simples pero, principalmente, gracias a su portabilidad de un sistema operativo a otro. Otra de las ventajas es que es un lenguaje interpretado, es decir, permite la prueba de código de una forma más rápida.

Para una mayor comprensión, debemos definir dos conceptos:

- *TUN/TAP,* que es una interfaz de red virtualizada.

- VETH/VLINK, el cual es un cable virtualizado que permite conectar dos puertos entre sí y además analizar el tráfico que circula en él.

Para desarrollar la práctica adquirimos los conocimientos del controlador Open vSwitch. Todos sus comandos los hemos enunciado en la sección de requisitos, y a continuación, los enumeramos y definimos de forma sencilla.

Los siguientes comandos los utilizamos para levantar una interfaz y permitir el tráfico del *firewall* para las máquinas y cables virtuales:

- **`ip link set <nombre>`** (Ej.: `vtrunk_0, br0, uml1.0`) **`up/down`** para encender o apagar las interfaces.
- **`iptables -I FORWARD -i <nombre>`** (Ej.: `uml+` es el que nosotros usamos) **`-o <nombre> -j ACCEPT`**
- **`iptables -I FORWARD -i <nombre>`** (Ej.: `vtrunk+`, el nombre que usamos para los *Ethernet* virtuales) **`-o vtrunk+ -j ACCEPT`**

Los comandos para generar la topología de red, que sirven para crear, eliminar y poner en modo STP un *bridge* virtual:

- **`ovs-ofctl add-br <nombre>`** (Ej.: `br0`) creamos un *bridge*.
- **`ovs-ofctl del-br <nombre>`** (Ej.: `br0`) eliminamos un *bridge*.
- **`ovs-ofctl set bridge <nombre>`** (Ej.: `br0`) **`stp_enable=true`** habilita el protocolo STP para evitar bucles entre *bridges,* debido al *broadcast storm*.

Los siguientes comandos sirven para agregar una interfaz y un cable virtual:

- **`ip tuntap add <nombre>`** (Ej.: `uml1.0`) **`mode tap`** añadir una interfaz virtual.

- **ip link add <nombre>** (Ej.: `vtrunk0`) **type veth peer name <nombre>** (Ej.: `vtrunk1`) creamos los cables virtuales para conectar nuestros *bridges.*

El siguiente comando sirve para conectar a un *bridge* un elemento:

- **ovs-vsctl add port <bridge> <elemento>** asignamos a un puerto del primer campo el dispositivo del segundo.

Comando que se utiliza para cambiar el número de puerto al que está conectado un elemento de un *bridge*:

- **ovs-vsctl set interface <nombre>** (Ej.: `uml1.0`) **ofport_request=<numero>** (Ej.: 2) cambia el número de puerto *OpenFlow*.

Comandos para mostrar datos importantes:

El siguiente comando muestra todas las conexiones de un bridge, **ovs-ofctl show <nombre>** (Ej.: `br0`).

Para mostrar una lista de todos los *bridges*, **ovs-vsctl list-br**. Para mostrar todos los elementos conectados a un *bridge*, **ovs-vsctl list-ports <nombre>** (Ej.: `br0`).

Con el siguiente comando mostramos el puerto *OpenFlow* al que está conectado un elemento a un *bridge*: **ovs-vsctl get interface <nombre>** (Ej.: `uml1.0`) **ofport**.

Comandos para el control de flujo. En este punto debemos explicar varias cosas importantes: el control de flujo se hace mediante un campo *match* y otro campo *actions.* El primero sirve para saber a qué afecta, y el segundo, qué acción toma:

- Criterios que pueden formar parte del campo *match*: **in_port**(puerto de entrada del bridge al que debe afectar), **dl_src**(*data_link source*, es la fuente del que proviene el paquete), **dl_dst, dl_vlan, dl_type, ip, arp, icmp, tcp, udp, ipv6, icmp6, tcp6, udp6.**

- Criterios que pueden formar parte del campo *actions:* **normal**, **flood, all, drop, resubmit, <puerto>, mod_vlan_vid, strip_vlan, mod_dl_src, mod_dl_dst, mod_nw_src, goto_table**.

Las reglas se pueden agrupar en tablas, los valores para identificarlas van desde 0 a 253 y se ordenan por prioridad 0 a 65536.

Con el siguiente comando mostramos los flujos definidos en un *bridge* hasta el momento: **`ovs-ofctl dump-flows <nombre>`** (Ej.: `br0`).

A continuación mostramos algunos ejemplos para agregar y eliminar flujos:

- **`ovs-ofctl add-flow br0 “priority=100 in_port=1 actions=resubmit(,101)”.`** Este comando permite redireccionar el tráfico que recibe el puerto 1 a la tabla 101 con prioridad 100.
- **`ovs-ofctl add-flow br0 “table=101 priority=0 actions=drop”.`** Este comando bloquea el tráfico de la tabla 101 con prioridad 0, es decir, la máxima.
- **`ovs-ofctl add-flow br0 “table=101 priority=100 ip,nw_dst=192.168.1.1 actions=normal”.`** Este comando permite un tráfico normal de los paquetes a la tabla 101, con destino a la IP asignada.
- **`ovs-ofctl del-flows br0 “in_port=1”.`** Elimina la regla `in_port=1` de `br0`.
- **`ovs-ofctl del-flows br0 “table=101”.`** Elimina la regla `table=101` de `br0`.

Comando para reglas QoS. Para asegurar la calidad del servicio, con los siguientes comandos creamos las colas con el máximo de ancho de banda que les proporcionamos:

- **`ovs-vsctl create queue other-config:max-rate=1000.`** Obtenemos `a1c74455-5cb4-4379-9bb2-b7e79b47edb8`.
- **`ovs-vsctl create queue other-config:max-rate=2000.`** Obtenemos `fb918a5b-4d3f-4613-a299-cd338831c2da`.

Estos comandos permiten la creación de una cola para asignar un máximo de ancho de banda: en el primer caso 100MB y en el segundo 200MB.

Ahora creamos la calidad del servicio de tipo HTB con un ancho de banda de 1Gbps, en el cual le asignamos las dos colas:

- **`ovs-vsctl create qos type=linux-htb \ other-config:max-rate=100000 \ queues:10=a1c74455-5cb4-4379-9bb2-b7e79b47edb8 \ queues:20=fb918a5b-4d3f-4613-a299-cd338831c2da`**. Obtenemos `3882a54d-859a-4f6f-9564-da31b36d9d5c`.

- **`ovs-vsctl set port eth0 qos=3882a54d-859a-4f6f-9564-da31b36d9d5c`**. Agregamos la calidad de servicio a la interfaz que está conectada a Internet.

Se crean los flujos para asignar las colas recién creadas:

- **ovs-ofctl add-flow br0 in_port=5,actions=set_queue:10,normal.**
- **ovs-ofctl add-flow br0 in_port=6,actions=set_queue:20,normal.**

El *mirroring* tiene como utilidad copiar el tráfico de uno o varios puertos de un bridge a un puerto de destino. A continuación mostramos varios ejemplos en los que creamos un nuevo puerto y lo agregamos al *bridge*.

En el primer caso copiamos todo el tráfico al nuevo puerto y en el segundo copiamos solo el de uno de los puertos:

- **ovs-vsctl add-port br0 uml5.0 -- \ --id=@p get port uml5.0 -- \ --id=@m create mirror name=m0 select-all=true output-port=@p -- \ set bridge br0 mirrors=@m**

- **ovs-vsctl --id=@p1 get port uml1.0 -- \ --id=@p get port uml3.0 -- \ --id=@m create mirror name=m0 \ select_dst_port=@p1 select_src_port=@p1 output-port=@p -- \ set bridge br0 mirrors=@m**

Por último, usamos el siguiente comando para limpiar todos los casos de *mirroring* sobre un *bridge*, utilizando el siguiente comando: **ovs-vsctl clear Bridge br0 mirrors**.

Para acabar, es relevante destacar la existencia de un problema en el que nos sitúa Tcl-Tk. Al no tener una POO (Programación orientada a objetos) tan desarrollada como otros lenguajes tales como Java o C++, provoca que tuviéramos que estudiar cómo poder orientar este lenguaje hacia una programación modularizada, conllevando así un estudio exhaustivo con el fin de desarrollar nuestra metodología.

## 4.2 Metodología

Hemos decidido desde el primer momento aplicar técnicas de metodología ágiles. Lo consideramos técnicas ya que al ser un único componente de equipo no nos parecía correcto seguir expresamente una, como por ejemplo Scrum, Crystal o la programación extrema (EX, *extreme programming*)

Cierto es que no ha habido división de poderes, es decir, no han sido asignados los roles de, por ejemplo, jefe de proyecto, programadores, comprobadores,... debido al problema anteriormente expuesto: Es un único componente y, por tanto, todo el trabajo recae en una misma persona. Además, se decidió junto al cliente, hacer una estimación de coste y tiempo de manera somera, ya que no se poseía de un histórico en el cual basarnos.

Por ello, decidimos basarnos en las ventajas que nos aportan las metodologías ágiles, empleando de ésta forma las siguientes técnicas:

- Crear una especificación de requisitos, teniendo presente, además que existe contacto constante con el cliente. Dicha especificación se puede iniciar sin tenerla totalmente cerrada, esto es, con la posibilidad de ir modificándose en el transcurso del desarrollo.
- Revisar los posibles riesgos que tendríamos a la hora de investigar los conocimientos necesarios, así como crear la documentación y la elaboración del código para que, de esta forma, podamos darnos cuenta a tiempo de dichos riesgos y ser capaces de contrarrestarlos.
- Poseer un sistema de control de versiones para poder volver en cualquier momento a versiones anteriores y tener la posibilidad de recuperarlo todo.
- Organizar reuniones constantes con el cliente, en las cuales se produce la retroalimentación del proyecto y nos permite encaminarnos a lo que nos demanda, evitando, de esta forma, la pérdida de tiempo en ir por el camino incorrecto.

A grandes rasgos, éstos fueron los pasos más lógicos que aplicamos para que nuestro proyecto siguiera adelante. Creemos que, gracias a la aplicación de estas técnicas, cubrimos los problemas principales, identificamos los requisitos y definimos los posibles problemas que podrían suceder a la hora del desarrollo de la misma. Por otra parte, cabe destacar que se ha realizado y seguido esta metodología desde un principio del proyecto, y se ha ejecutado de una forma profesional que nos ha permitido definir un rumbo específico.

## 4.3 Especificación de requisitos

La especificación de requisitos *software* es donde identificamos el problema y las necesidades del cliente para así poder empezar a desarrollar nuestro proyecto. Una vez tuvimos claros los conceptos básicos de SDN, OpenFlow, Open vSwitch decidimos generar este documento. La elaboración se hizo de una forma concisa en las reuniones con el cliente. Principalmente, se le preguntaba cuáles eran las características que debía tener la aplicación y poco a poco se fueron identificando. Dichas características están abiertas a cambios durante todo el proyecto. Después de una serie de reuniones, decidimos que nuestra aplicación tendrá tres módulos, los cuales explicamos a continuación:

- **Topología**: este módulo se encargará de todas las funciones relacionadas con al creación de la red: agregar, eliminar, mostrar, guardar y cargar *bridges*, puertos, interfaces virtuales tipo TUN/TAP, interfaces físicas, cables

virtuales tipo VETH y la asignación de los puertos del *bridge*. Las funciones de este módulo son:

- **Añadir puente:** permite agregar un puente (*bridge*) donde conectaremos nuestras máquinas virtuales.
- **Añadir interfaz:** añade una interfaz virtual para las máquinas.
- **Añadir puerto:** añade un elemento al puerto de un *bridge*.
- **Añadir cable:** añade un cable virtual para conectar los puentes, (VETH).
- **Eliminar puente:** permite eliminar un puente de nuestra topología de red.
- **Eliminar interfaz:** elimina una interfaz virtual.
- **Eliminar puerto:** elimina un puerto de un *bridge*.
- **Eliminar cable:** elimina un cable virtual.
- **Mostrar:** muestra los *bridges* y los elementos conectados a los puertos.
- **Guardar:** recoge el estado actual del controlador y lo traduce a un formato *Json* para después pasarlo a la función de guardar del fichero.
- **Cargar:** recibe un estado guardado mediante la lectura de un fichero *Json* previamente creado y lo carga en la aplicación.
- **Limpiar:** elimina toda la configuración del controlador y la memoria.

- **Control**: este módulo se encarga del control de flujo de los *bridges*, permitir o denegar un flujo, guardarlos, cargarlos y mostrarlos. Las funciones de este módulo son:

  - **Añadir flujo:** agrega controles de flujo al *bridge* seleccionado.
  - **Eliminar flujo:** elimina los controles de flujo del *bridge* seleccionado.
  - **Mostrar:** muestra los flujos del *bridge* seleccionado.
  - **Guardar flujo:** guarda el control de flujo del *bridge* seleccionado.
  - **Cargar flujo:** carga un estado de control de flujo en un *bridge* existente.
  - **Limpiar:** limpia el control de flujos de un *bridge*.

- **Archivos**: este módulo se encarga de guardar en un fichero *Json* el estado actual del controlador, tanto la topología como el control de flujo. Las funciones de este módulo son:

  - **Guardar Json:** Guarda un fichero cuyo nombre y contenido recibirá.
  - **Cargar Json:** Lee el contenido de un archivo y devuelve el contenido.
  - **Guardar control de flujo:** Guarda el estado del control de flujo de un *bridge*.

Después de identificar exactamente los módulos que necesitamos y valorar cuáles eran las funciones que debía tener cada uno, se decidió dividir la aplicación en tres capas para conseguir las características anteriormente comentadas:

- **Capa de presentación**: en la cual se encuentra el controlador de la aplicación y la parte gráfica con la cual interactúa el usuario.

- **Capa de negocio**: donde se encuentra toda la carga de procesamiento de datos de la aplicación.

- **Capa de datos**: en la cual podemos acceder a la base de datos de Open vSwitch o a los archivos que genera nuestra aplicación

Estas tres capas nos ofrecen un código más claro y mantenible, ya que lo que buscamos es que nuestro proyecto pueda seguir siendo desarrollado en un futuro por nuevos estudiantes.

A continuación vamos a explicar como se ha desarrollado la aplicación. Mostraremos unos ejemplos del código para entender mejor cómo se ha realizado cada una de las partes y posteriormente en el manual de usuario enseñaremos cada una de las vistas y su utilización.

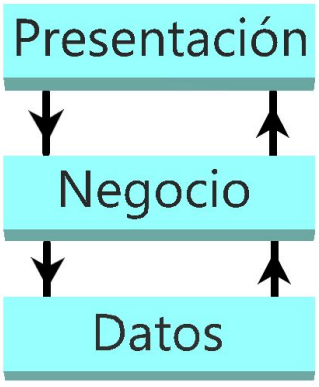


Figura 4.1 Arquitectura de la aplicación.

Como se representa en la figura 4.1, la aplicación está dividida en las tres capas anteriormente comentadas. Estas capas pueden comunicarse entre ellas y cada una tiene un objetivo.

### 4.3.1 Capa de presentación

En esta capa encontramos el controlador que comunica el servicio de aplicaciones con las vistas. De una forma sencilla, es el encargado de captar lo que le comunica el usuario y transmitirlo a la parte lógica del programa. Además, se encuentran las vistas con la parte gráfica de la aplicación, y también se muestran al usuario para que éste pueda interactuar con la misma. A continuación, vamos a mostrar un ejemplo visual del código de las vistas y del controlador.

```
oo::class create controller {
    constructor {} {
    my variable businessTopology
    my variable businessControl
    set businessTopology [businessTopology new]
    set businessControl [businessControl new]
}
    method action {event data} {
    my variable businessTopology
    my variable businessControl

    switch $event {
          SA_addBridge {
    set result [$businessTopology addBridge $data]
    return $result
     }
```

Figura 4.2 Controlador de la aplicación

En la figura 4.2 mostramos el controlador su función consiste en recibe un evento y un dato, desde las vistas y comunicarse con la capa inferior para procesar los datos, una vez realizado esto lo devuelve a las vistas y muestra los resultados al usuario.

```
    set m .menu.files
    menu $m -tearoff 0
    .menu add cascade -label "Archivo" -menu $m -underline 0
    $m add command -label "Guardar topologia" -command {destroy .panel;
    windowSaveTopology}
    $m add command -label "Cargar topologia" -command {destroy .panel;
    windowLoadTopology}
```

```
      $m add command -label "Guardar control" -command {destroy .panel;
      windowSaveFlows}
       $m add command -label "Cargar control" -command {destroy .panel;
      windowLoadFlows}
```

Figura 4.3 Vista principal.

En la figura 4.3 mostramos un cachito del código simplemente para explicar que es una ventana en la cual mostramos todas las funciones de nuestra aplicación mediante menús, una vez que vamos visitando cada uno de ellos se muestran la vista de cada función.

```
      proc event_X {}
      proc window_X {}
```

Figura 4.4 Ejemplo de formato de vistas.

En la figura 4.4 mostramos la forma de crear una vista. Todas ellas están elaboradas con el mismo sistema una función *event,* la cual se comunica con el controlador enviando unos datos para procesar la petición del usuario, y una función *window* la cual muestra los componentes gráficos de la vista tal como hemos visto en la figura 4.3.

### 4.3.2 Capa de negocio

Esta capa es la encargada de recibir una petición desde la capa de presentación y procesar los datos para devolver un resultado. Ha de tener también acceso a la capa de datos.

```
source dataAccess/dao_topology.tcl
source dataAccess/dao_file.tcl

oo::class create businessTopology {
  constructor {} {
       my variable daoTopology
       my variable daoControl
       my variable daoFile
       set daoTopology [daoTopology new]
       set daoControl [daoControl new]
       set daoFile [daoFile new]
  }

  method addBridge {bridge} {}
  method addInterface {interface} {}
  method addPort {from to} {}
  method addVlink {from to} {}
  method delBridge {bridge} {}
```

```
  method delPort {from to} {}
  method delVlink {from} {}
  method showInfo {} {}
  method saveTopology {fileName} {}
  method saveTopologyOld {fileName} {}
  method loadTopology {fileName} {}
  method loadTopologyOld {fileName} {}
  method cleanTopology {} {}
}
```

Figura 4.5 Capa de negocio, topología.

En la figura 4.5 mostramos el módulo de topología de la capa de negocio. En él se pueden ver todas las funciones de las que hablamos en el apartado de especificación de requisitos.

### 4.3.3 Capa de acceso a datos

Esta capa es la encargada de comunicarse con la base de datos de Open vSwitch y con los ficheros que creamos para guardar y cargar el estado de nuestra aplicación.

```
oo::class create daoTopology {
  method addBridge {bridge} {
       if {[catch {exec sudo ovs-vsctl add-br $bridge} errmsg]} {
       return 0
       }
       return 1
  }

  method addInterface {interface} {}
  method addVlink {from to} {}
  method addPort {from to} {}
  method addPorta {from to} {}
  method setOfPort {interface ofPort} {}
  method upElement {element} {}
  method delBridge {bridge} {}
  method delInterface {interface} {}
  method delPort {from to} {}
  method delVlink {from} {}
  method downElement {element} {}
  method listBr {} {}
  method listPort {bridge} {}
  method getInterfaceStatus {interface} {}
  method getVethConnection {interface} {}
```

```
  method existInterface {interface} {}
  method getOfPort {interface} {}
}
```

Figura 4.6 Capa de acceso a datos, topología.

En la figura 4.6 mostramos la capa de acceso a datos del módulo de topología. Como ya comentamos antes, está totalmente basada en la especificación de requisitos cumpliendo con todas las necesidades.

## 4.4 Plan de gestión de riesgos

En este apartado hablaremos del plan de gestión de riesgos que hemos realizado en nuestro proyecto, así como los dos riesgos más graves que podrían afectar a nuestra aplicación.

Lo primero que hicimos fue analizar cuáles eran los posibles riesgos que podríamos encontrarnos a la hora de realizar el proyecto, desde la investigación, a la elaboración del código: obtuvimos diez riesgos. Una vez identificados los clasificamos de dos formas, la primera la probabilidad que se diera y la segunda la gravedad de que sucediera.

De esta forma, mediante las tablas de SQAS-SEI (Normas de certificación) obtenemos el impacto que podrían producir en nuestro trabajo y los ordenamos de mayor a menor.

Nos basamos en el principio de Pareto, el cual dice que el 80% del riesgo real se encuentra en el 20% de los analizados, por lo que cogimos dos riesgos más uno para elaborar un plan en el caso de que se volvieran realidad. Son los siguientes:

### (R02): Atraso en las entregas.

Es el principal riesgo que revisamos desde el principio. El problema del proyecto es que se realiza por una única persona, por lo tanto existen mayores dificultades ya que no puede recibir ayuda de un compañero respecto a los lenguajes, memoria, creación de código, etc. Por lo que es evidente que puede ocasionar estos atrasos. La solución será identificar las semanas de mayor trabajo para evitar que coincida con las demás asignaturas para así, de esta forma, obtener mayor tiempo y poder cumplir con las entregas del profesor.

### (R03): Desconocimientos de las tecnologías necesarias.

Otro de los riesgos más serios que valoramos fue el problema de a qué tecnologías nos enfrentamos. El conocimiento de las SDN y de Tcl-Tk es totalmente nulo, por lo que es necesario para evitar este problema estudiarlos en profundidad. De esta forma, invirtiendo unos meses y gracias a la ayuda del profesor y a nuestra propia investigación, pudimos superar este escalón.

### (R04): No disponer del tiempo necesario para el proyecto

Como comentamos en el riesgo “R02”, uno de los problemas evidentes es que el proyecto es desarrollado por una única persona, por lo que dificulta mucho el trabajo. Este riesgo trata de la probabilidad que tenemos de no llegar a tiempo en la entrega por no poder acabar nuestro proyecto. La solución a este riesgo es identificar la fecha de entrega, hablar con el profesor sobre una entrega realista que no se salga de nuestras expectativas y lógicamente aumentar las horas de trabajo. No podemos llegar a este punto: El proyecto debe entregarse.

## 4.5 Control de versiones

Para evitar el riesgo de la pérdida de datos, tanto de la documentación como el código desarrollado, hemos realizado todo el proyecto mediante dos plataformas: La primera, para los documentos ha sido *Google Drive,* donde están almacenados cada uno de los cambios. La segunda para el código, la cual ha sido Github [9] donde se ha actualizado el código mientras se finalizaban los módulos. Lo mostramos en la Figura 4.7.

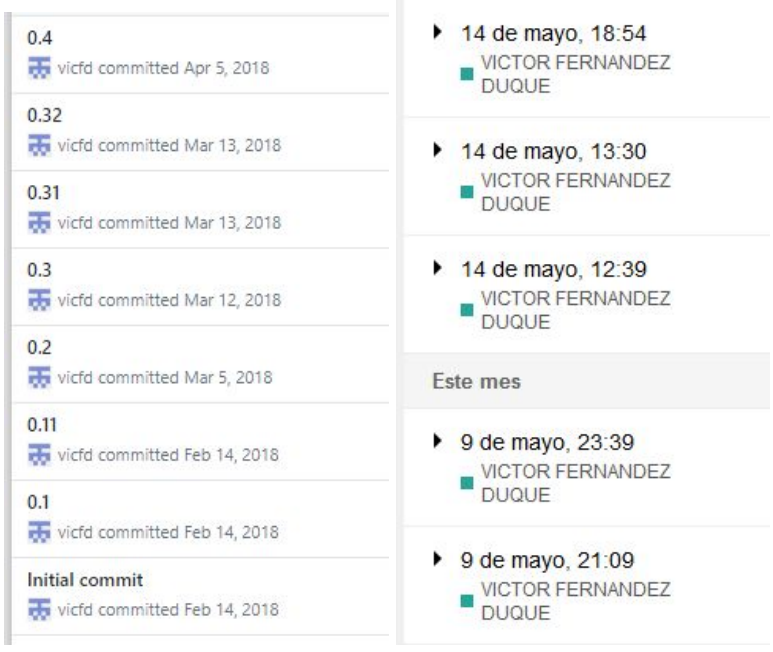


Figura 4.7 Control de versiones, Github y Google Drive.

## 4.6 Planificación

La planificación la hemos desarrollado mediante el programa *Planner* en Debian, mediante el cual hemos realizado una pequeña guía de las tareas que debíamos hacer en unos tiempos marcados, para así evitar retrasos en las entregas, tal como mostramos en la Figura 4.8.

| WBS | Nombre | Inicio | Fin | Trabajo | Duración | Asignado a | % Completado |
|---|---|---|---|---|---|---|---|
| 1 | Estudio requisitos | oct 2 | nov 24 | 40d | 40d | Víctor | 100 |
| 2 | Planificacion | nov 27 | dic 22 | 20d | 20d | Víctor | 100 |
| 3 | Descanso por exámenes | dic 25 | feb 2 | 30d | 30d | Víctor | 100 |
| 4 | Programación | feb 5 | abr 6 | 45d | 45d | Víctor | 95 |
| 5 | Memoria | abr 9 | may 24 | 34d | 34d | Víctor | 80 |
| 6 | Entrega | may 25 | may 25 | 1d | 1d | Víctor | 0 |

Figura 4.8 Planificación, Planner

# 5. Manual de usuario

En este apartado mostraremos toda la aplicación y cuál es su forma de uso. Iremos paso a paso por todas las funciones enseñando además un ejemplo gráfico para poder guiarnos.

## 5.1 Vista principal

En la figura 5.1 mostramos la vista principal de nuestra aplicación. Como se puede ver tiene tres partes, el menú con todas las funciones, la posición central donde iremos mostrando cada una de las vistas y el cuadro de texto que muestra las notificaciones del programa.


Figura 5.1 Vista principal del controlador.

## 5.2 Menú archivo

En este apartado comentaremos el menú de archivo. Encontramos cuatro funciones: guardar topología, cargar topología, guardar control, y cargar control tal como mostramos en la figura 5.2.


Figura 5.2 Menú archivo.

En la figura 5.3 mostramos la vista de guardar topología. Nos aparece un cuadro de texto en el cual debemos poner el nombre de la carpeta en la que queremos guardar el estado actual del programa.


Figura 5.3 Vista guardar topología.

En la figura 5.4 muestra el resultado al pulsar el botón de “Guardar topología”. La aplicación ha guardado dentro de la carpeta *save*, en el directorio principal de la aplicación, una nueva carpeta llamada “ejemplo” en la cual se encuentran los datos del estado actual de la topología y cada uno de los flujos de control de los *bridges* virtuales.

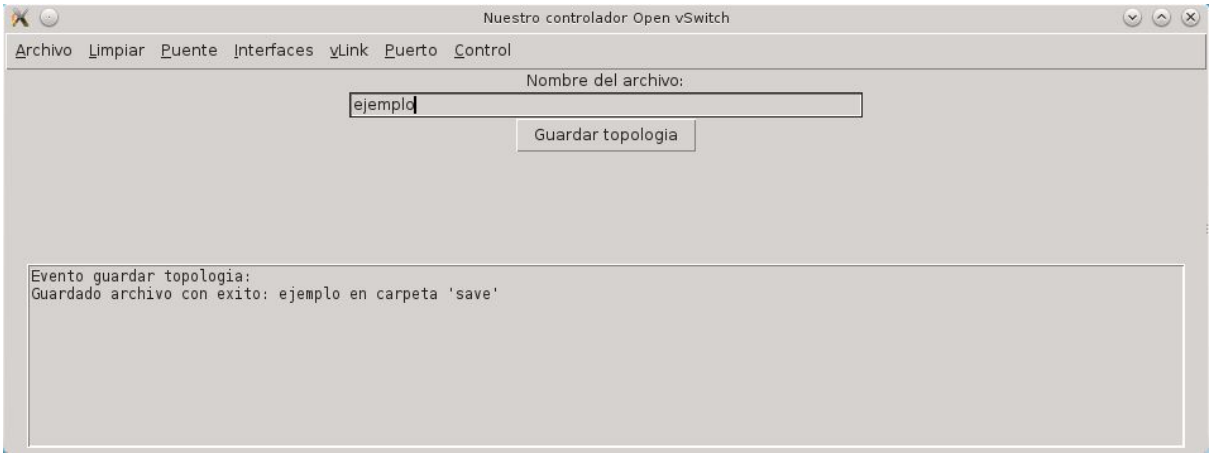


Figura 5.4 Vista guardar topología, resultado.

En las figuras 5.5 y 5.6 mostramos el contenido que genera el guardar el estado de la aplicación. En la primera vemos que ha creado tres documentos los que empiezan por “br” son los estados guardados para el control de flujo de cada uno de los *bridges* virtuales y el save es el que contiene la topología de red, el cual es un fichero *Json* del que podemos ver su formato en la segunda imagen.


Figura 5.5 Contenido del guardado.

```
"topologia":
{
    "veth" : [{"veth1.0": "veth1.1"}],
    "tuntap" : ["uml1.0", "uml2.0", "uml2.1"],
    "bridge" : [{"br0" : ["br0.ncovs", {"uml1.0": 1}, {"uml2.0": 2},
   {"veth1.0": 3}]},   {"br1" : ["br1.ncovs", {"uml1.1": 1}, {"uml2.1": 2},
   {"veth1.1": 3}]}]
}
```

Figura 5.6 Contenido de save.ncovs.

En la figura 5.7 mostramos la vista de cargar para buscar un fichero donde hayamos guardado anteriormente el estado del programa y posteriormente cargarlo.


Figura 5.7 Vista cargar topología.

En la figura 5.8 podemos observar lo que ocurre al seleccionar el archivo, nos muestra dónde está situado y además da como resultado que se ha cargado correctamente. Después de este comando es recomendable ir al menu “Puente” y utilizar la función “Mostrar puentes” para comprobar que se ha realizado correctamente.


Figura 5.8 Vista cargar topología, resultado.

En la figura 5.9 mostramos la vista de guardar control de flujo, almacena el estado actual de un *bridge* en un fichero que defina el usuario. Se desplegará un menú para introducir el nombre del archivo con el que queremos guardarlo.


Figura 5.9 Vista guardar control.

En la figura 5.10 podemos observar el resultado que nos ofrece al guardar el control de flujo del *bridge* “br1” y nos muestra el fichero donde lo hemos guardado. El contenido del mismo sería:

```
cookie=0x0, duration=44.593s, table=0, n_packets=34 n_bytes=4854, idle_age=12
priority=0 actions=normal
```

Donde podemos ver que el único flujo existente es tráfico normal con prioridad cero.


Figura 5.10 Vista guardar control, resultado.

En la figura 5.11 mostramos la vista de cargar flujo. Una vez hemos guardado el control de flujo de uno de nuestros *bridges*, podemos seleccionar el archivo para volver al estado anterior. Además deberemos introducir por texto o por cuadro de diálogos el nombre del *bridge* en el cual queremos volcar el archivo.


Figura 5.11 Vista cargar control.

En la figura 5.12 mostramos el resultado que nos ofrece el cargar el control de flujo sobre el *bridge* “br1”. El cuadro de diálogo nos muestra que se ha hecho de forma correcta sobre el *bridge* introducido y qué fichero hemos usado.

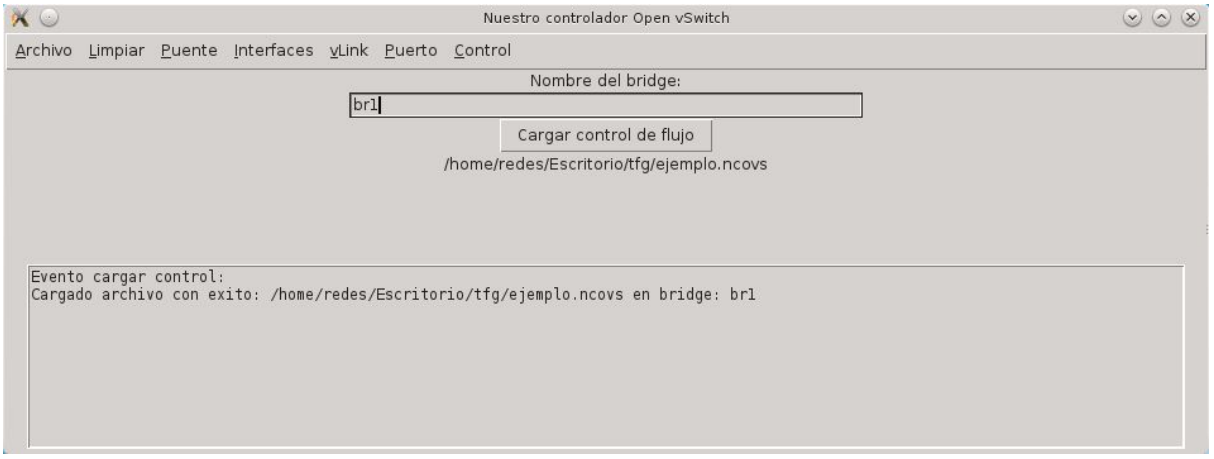


Figura 5.12 Vista cargar control, resultado.

## 5.2 Menú limpiar

En este apartado mostramos las funciones utilizadas para eliminar el estado actual de la aplicación o el control de flujo de un *bridge* tal como mostramos en la figura 5.13.


Figura 5.13 Menú limpiar.

En la figura 5.14 vemos el evento que limpia toda la topología de red que tenemos actualmente. Ésta función no tiene parte visual ya que sólo requiere de la acción humana nada más invocarla. El resultado que ofrece es que se ha realizado correctamente por lo que ha borrado todos los *bridges* virtuales de la aplicación.


Figura 5.14 Evento limpiar topología.

En la figura 5.15 mostramos la vista de limpiar el control de flujo de un *bridge*. Sólo necesitamos el nombre de éste para limpiar sus reglas de control de flujo.


Figura 5.15 Vista limpiar control de flujo.

En la figura 5.16 mostramos el resultado de limpiar el control de flujo del *bridge* "br1" dice que se ha realizado con éxito. Por lo tanto, si usamos la función de control "Mostrar flujo" en el mismo *bridge,* ésta nos dirá que no tiene flujos de control debido a que los hemos eliminado totalmente.


Figura 5.16 Vista limpiar control de flujo, resultado.

## 5.3 Menú puente

En este apartado hablaremos del menú puente, en el cual tenemos las funciones de crear, eliminar y mostrar los *bridges* y sus puertos, tal como podemos ver en la figura 5.17.


Figura 5.17 Menú puente.

En la figura 5.18 mostramos la vista para crear un *bridge* virtual. Lo que debemos hacer es introducir un nombre para crearlo.


Figura 5.18 Vista crear bridge.

En la figura 5.19 mostramos el resultado al pulsar en crear puente. Introducimos como nombre “br0”, por lo que tendremos en nuestra topología un *bridge* virtual con ese nombre.


Figura 5.19 Vista crear bridge, resultado.

En la figura 5.20 mostramos la vista para la eliminación de un *bridge* virtual. Deberemos introducir el nombre del *bridge* que queremos borrar y pulsar sobre “eliminar puente”.


Figura 5.20 Vista eliminar *bridge*.

En la figura 5.21 mostramos el resultado al pulsar sobre el botón eliminar puente. Obtenemos que se ha borrado con éxito por lo que el *bridge* virtual desaparece de la base de datos de Open vSwitch.

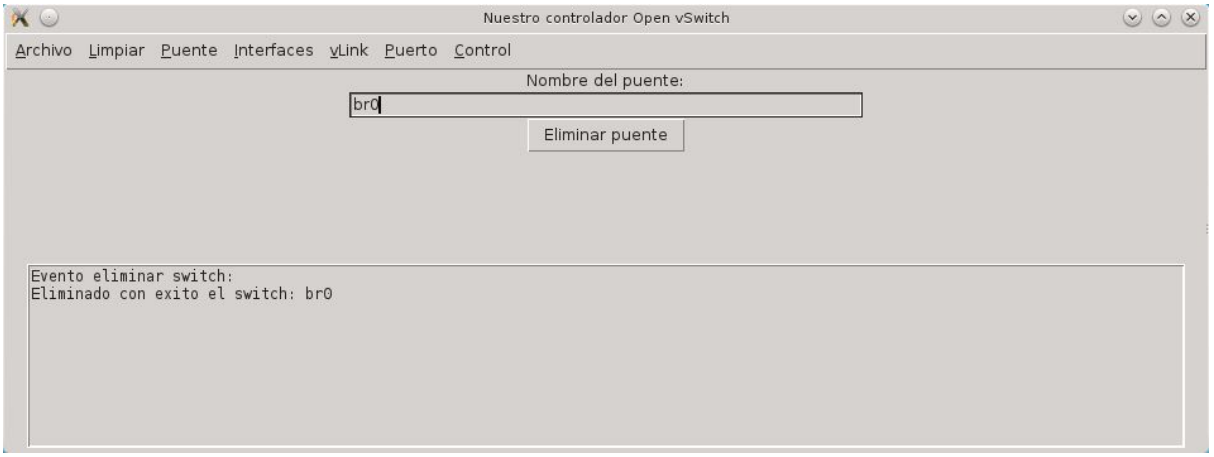


Figura 5.21 Vista eliminar *bridge*, resultado.

En la figura 5.22 se puede observar el evento “mostrar puentes”. Éste nos devuelve cada uno de los *bridges* virtuales que tenemos y los componentes conectados a nuestros puertos, no posee una vista debido a que es una función directa, como mencionamos anteriormente en el caso de “limpiar topología”, ya que no necesita entradas de datos.


Figura 5.22 Evento mostrar bridge y puertos.

## 5.4 Menú interfaces

En este apartado hablaremos del menú de interfaces. Tenemos las funciones agregar y eliminar interfaz, con las cuales podemos crear las interfaces de red para conectar nuestras máquinas virtuales, tal como mostramos en la figura 5.23.


Figura 5.23 Menú interfaces.

En la figura 5.24 mostramos la vista de para agregar una interfaz de red TUN/TAP, para conectar nuestras máquinas virtuales. Debemos escoger un nombre y pulsar sobre “crear interfaz”.


Figura 5.24 Vista crear interfaces.

En la figura 5.25 observamos el resultado que nos ofrece al pulsar sobre “crear interfaz”. Se genera la interfaz virtual, que en nuestro caso hemos escogido “uml1.0”.


Figura 5.25 Vistar crear interfaces, resultado.

En la figura 5.26 mostramos la vista para eliminar nuestras interfaces virtuales. Debemos introducir el nombre de la que queramos borrar y pulsar sobre “eliminar interfaz”.


Figura 5.26 Vista eliminar interfaces.

En la figura 5.27 mostramos el resultado al eliminar la interfaz creada con anterioridad. En nuestro caso volvemos a introducir “uml1.0” y el programa nos dice que se ha realizado con éxito.

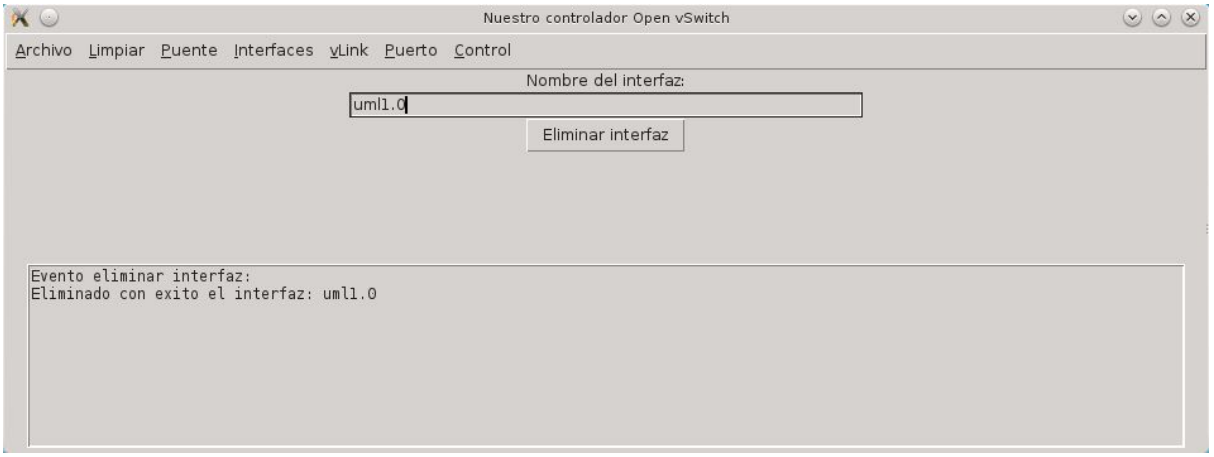


Figura 5.27 Vista eliminar interfaces, resultado.

## 5.5 Menú vLink

En este apartado hablaremos del menú vLink, el cual usamos para crear los cables virtuales para conectar nuestros *bridges*. Tenemos dos funciones: agregar y eliminar vLink, tal y como mostramos en la figura 5.28.

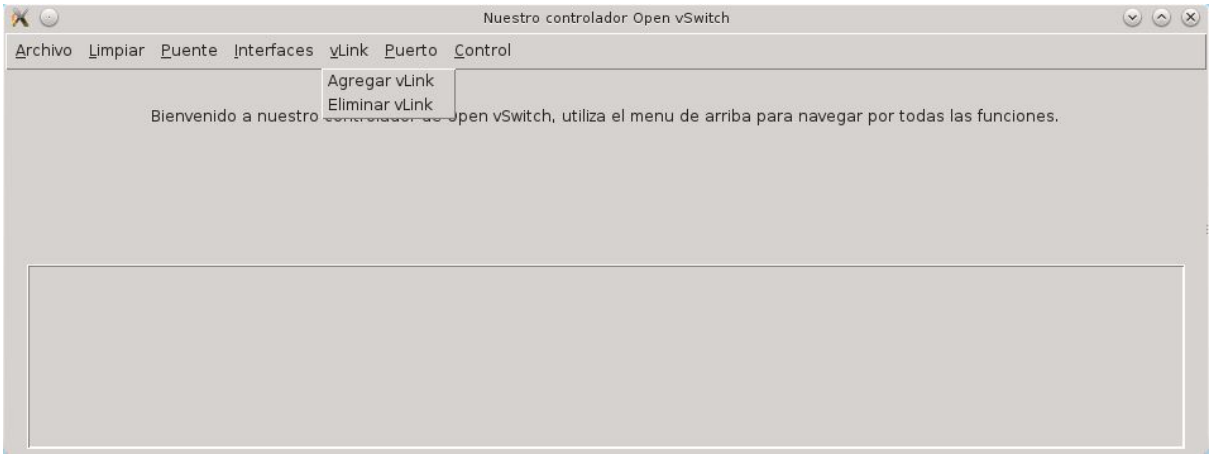


Figura 5.28 Menú vLink.

En la figura 5.29 mostramos la vista para crear un cable virtual para conectar dos *bridges* virtuales. Para ello necesitamos el nombre de los dos extremos y pulsar sobre el botón “crear vLink”.


Figura 5.29 Vista crear vLink.

En la figura 5.30 mostramos el resultado al pulsar sobre “crear vLink”, nos dice que se ha creado un cable virtual con dos extremos “vlink1.0” y “vlink2.0”.


Figura 5.30 Vista crear vLink, resultado.

En la figura 5.31 mostramos la vista para eliminar un cable virtual. Para realizar esta función sólo necesitamos eliminar uno de los extremos y con éste se eliminan los dos, por lo tanto, debemos introducir uno de ellos y pulsar sobre el botón “eliminar vLink”.


Figura 5.31 Vista eliminar vLink.

En la figura 5.32 mostramos el resultado al pulsar sobre el botón “eliminar vLink” hemos introducido uno de los extremos, “vlink1.0”, y nos dice que se ha eliminado satisfactoriamente.


Figura 5.32 Vista eliminar vLink, resultado.

## 5.6 Menú puerto

En este apartado hablaremos del menú puerto, el cual se utiliza para agregar y eliminar componentes a un *bridge* virtual. Disponemos de dos funciones: agregar y eliminar puerto tal como mostramos en la figura 5.33.

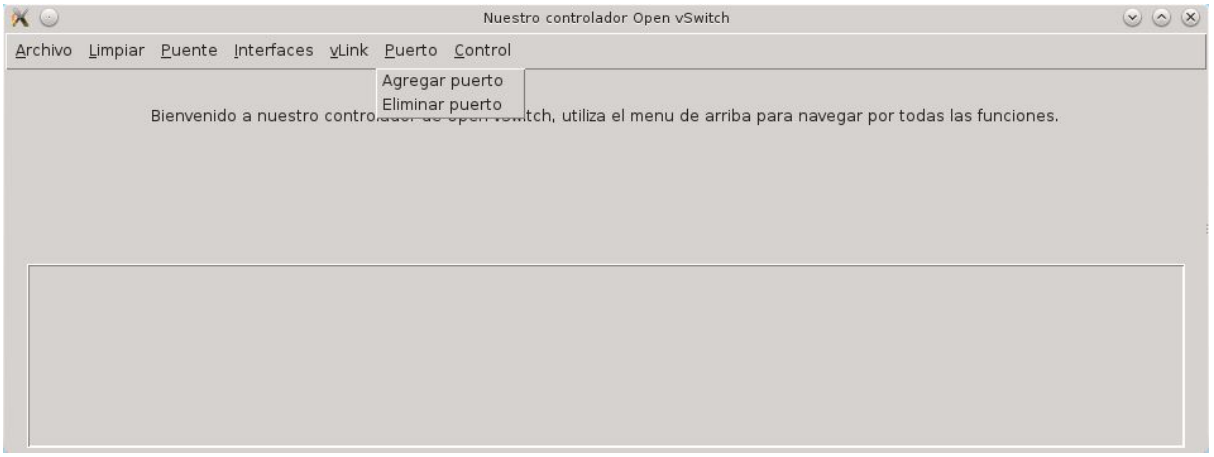


Figura 5.33 Menú puerto.

En la figura 5.34 mostramos la vista de agregar puerto, necesitamos el nombre de un *bridge* y el componente que queremos agregar, pulsamos en el botón “crear puerto” para ejecutar el evento.


Figura 5.34 Vista agregar puerto.

En la figura 5.35 observamos el resultado que nos ofrece al pulsar en el botón “crear puerto”. Hemos introducido el *bridge* “br1” y la interfaz “uml1.0”, nos muestra un mensaje que todo se ha realizado correctamente.

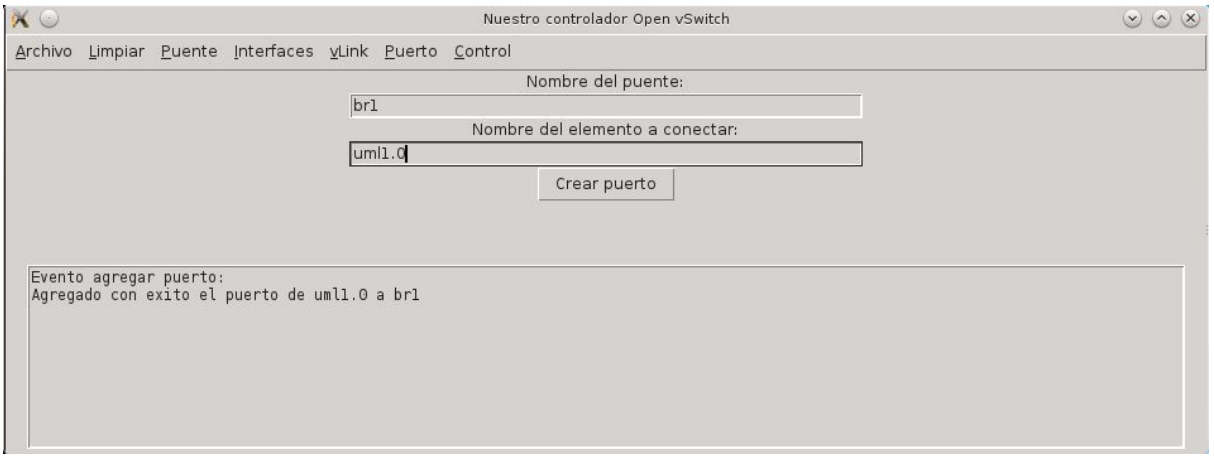


Figura 5.35 Vista agregar puerto, resuelto.

En la figura 5.36 mostramos la vista de eliminar un puerto a un *bridge*. Para ello necesitamos introducir el nombre del *bridge* y el componente que queremos desconectar del mismo, por último pulsaremos en el botón “eliminar puerto”.

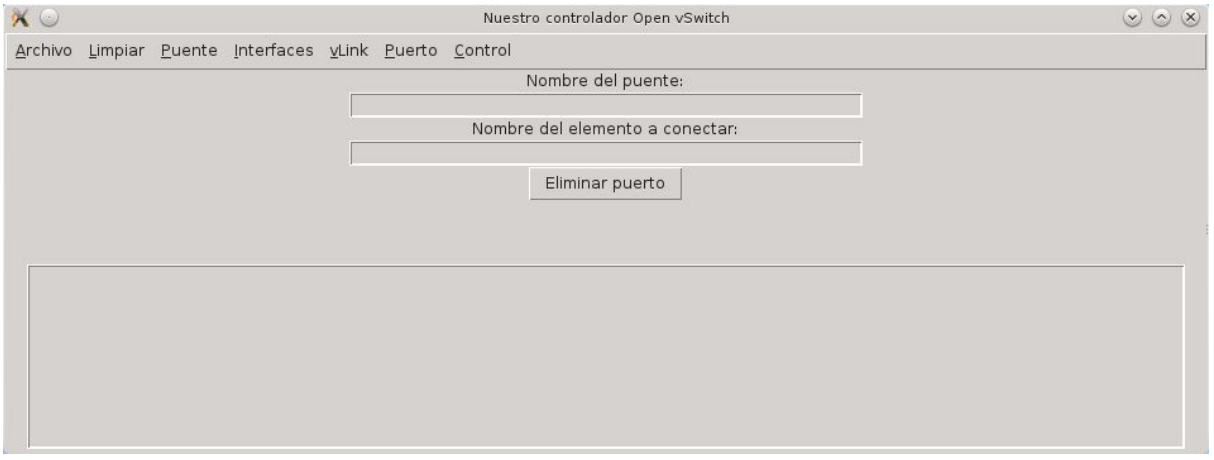


Figura 5.36 Vista eliminar puerto.

En la figura 5.37 mostramos el resultado que recibimos al pulsar en el botón “eliminar puerto”, hemos introducido el *bridge* “br1” y el componente “uml1.0”, nos muestra un mensaje de que se ha desconectado correctamente.

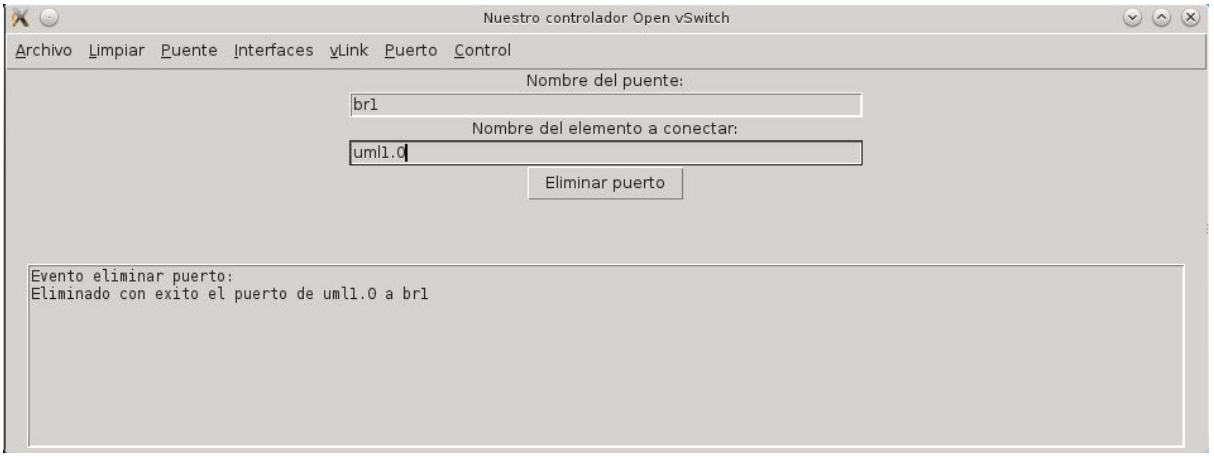


Figura 5.37 Vista eliminar puerto, resultado.

## 5.7 Menú control

En este apartado hablaremos del menú control, en el cual se puede modificar el control de flujo de nuestros *bridges* virtuales. Para ello poseemos las funciones agregar, eliminar y mostrar control de flujo, como se puede ver en la figura 5.38.

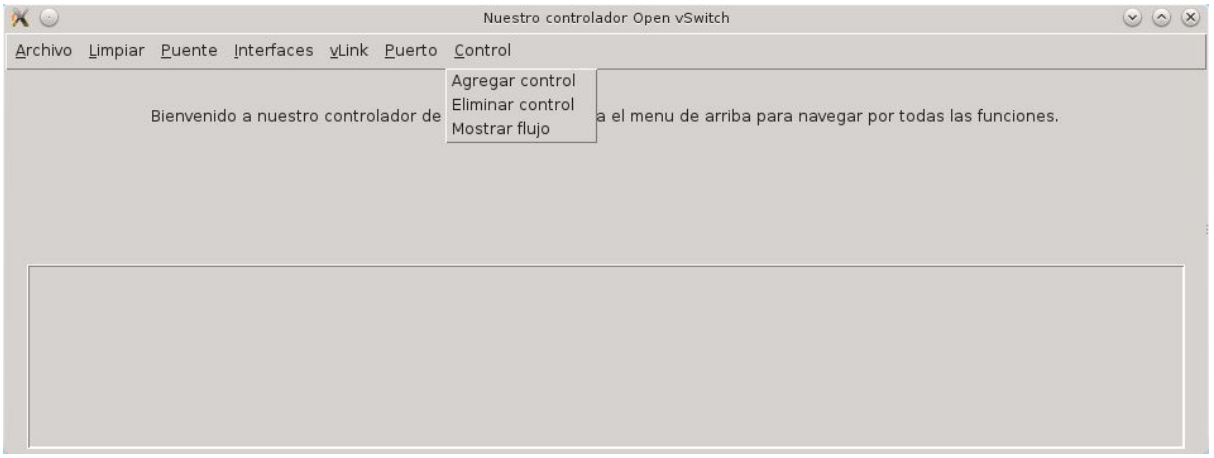


Figura 5.38 Menú control.

En la figura 5.39 mostramos la vista para agregar un control de flujo a un *bridge*. Para ello necesitamos el nombre del *bridge* y el flujo que queremos agregarle. Nos podemos guiar del apartado 2 y 3 de la memoria, donde explicamos cómo se configura el flujo. Una vez introducidos pulsamos en el botón “crear flujo”.


Figura 5.39 Vista agregar control.

En la figura 5.40 mostramos el resultado al pulsar sobre el botón “crear flujo”. Se ha añadido el *bridge* “br1” y un flujo de ejemplo el cual tiene prioridad 100, que afecta a la tabla 101, y que tiene como funcionalidad no aceptar ningún tráfico. Por último indica que la operación se ha realizado con éxito.

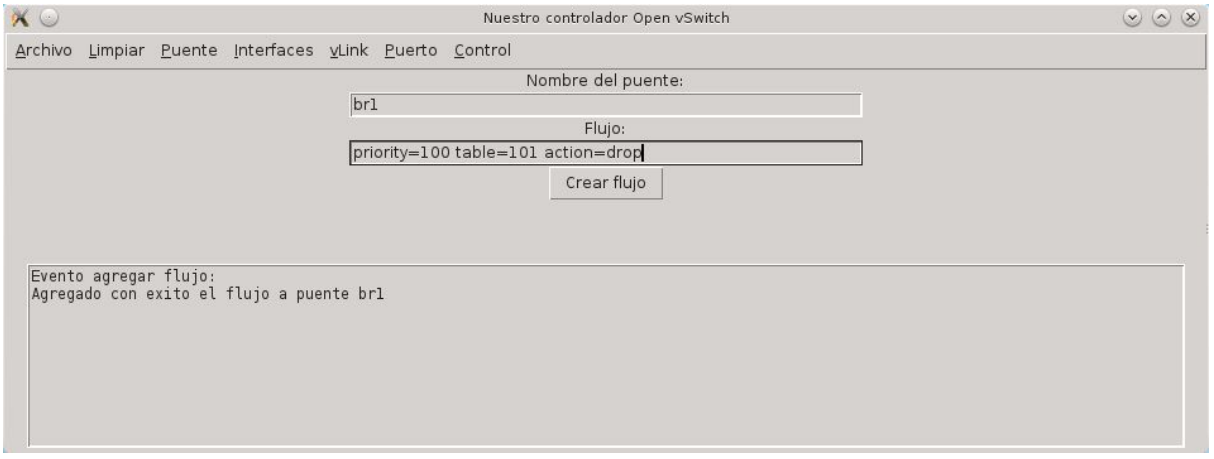


Figura 5.40 Vista agregar control, resultado.

En la figura 5.41 mostramos la vista para eliminar un flujo. Para éste fin, necesitamos introducir el nombre de un *bridge* y la parte *match* del flujo introducido, que se explica en el apartado 3 de la memoria, en la sintaxis empleada. Después pulsaremos sobre el botón “eliminar flujo”.


Figura 5.41 Vista eliminar control.

En la figura 5.42 mostramos el resultado que obtenemos al pulsar sobre el botón “eliminar control”. Hemos introducido el *bridge* “br1” y en flujo la parte match “table=101”. Nos indica mediante mensaje que se ha eliminado correctamente.

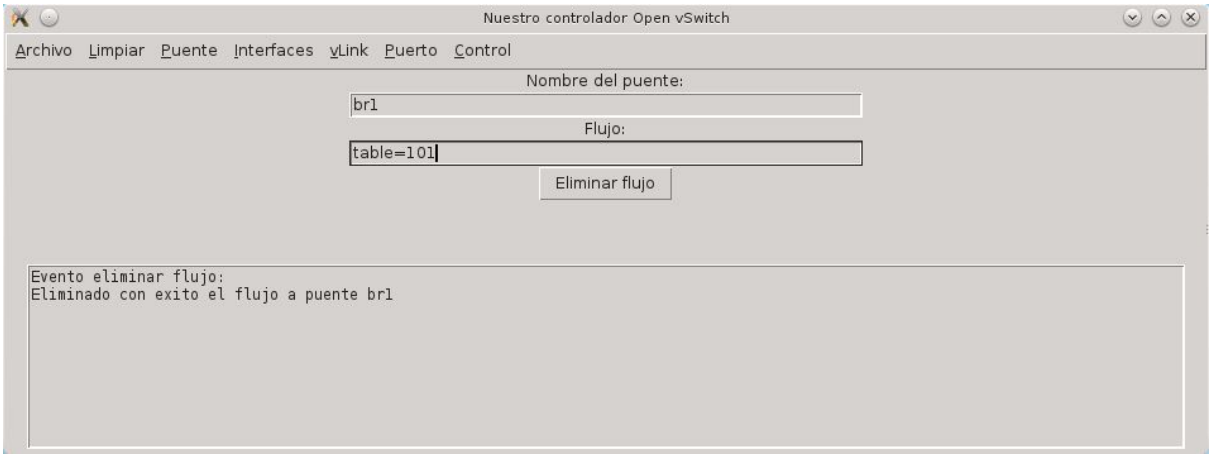


Figura 5.42 Vista eliminar control, resultado.

En la figura 5.43 observamos la función mostrar flujo, en la cual debemos introducir el nombre de un *bridge* y pulsar en el botón “mostrar flujo”.


Figura 5.43 Vista mostra control.

En la figura 5.44 mostramos el resultado que obtenemos al pulsar sobre el botón “mostrar flujo” introduciendo en el campo de texto el *bridge* “br1”. Indica que tiene un control de flujo que es “prioridad cero tráfico normal”.

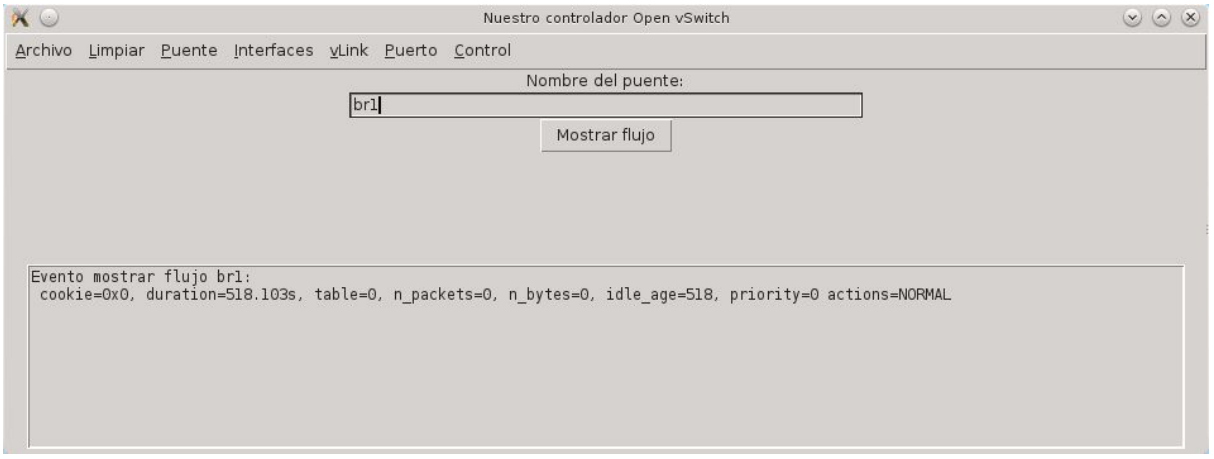


Figura 5.44 Vista mostra control, resultado.

## 5.8 Caso de uso

En este apartado haremos un caso de uso para demostrar cómo funciona nuestra aplicación mediante un ejemplo.

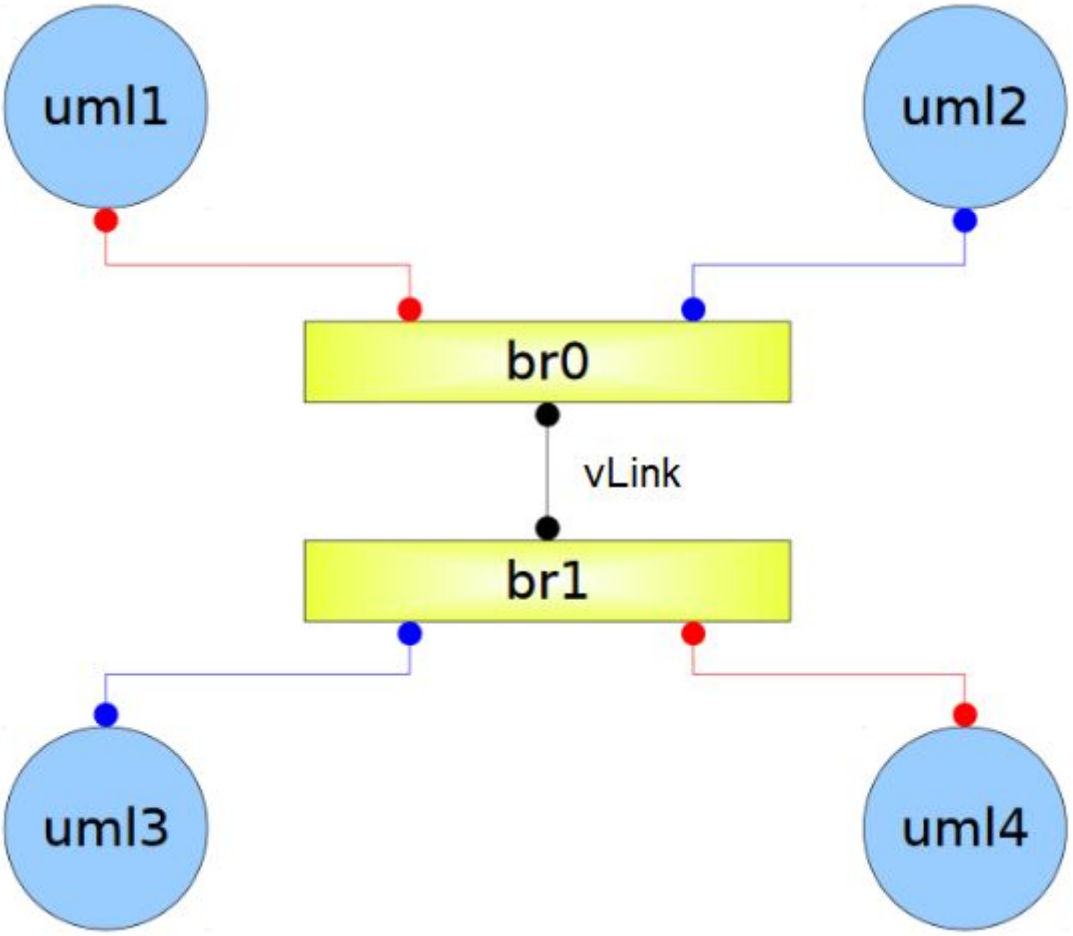


Figura 5.45 Caso de uso.

En la figura 5.45 mostramos la topología de red que vamos a crear mediante nuestra aplicación. Para ello debemos crear dos *bridges*, un enlace *vLink* y cuatro interfaces de red virtual para las máquinas UML para así, por último, conectarlos.

Para crear los *bridges* iremos al menú “Puente” y seleccionar la opción “Agregar puente”. Para crearlos simplemente debemos introducir los nombres y después pulsar sobre el botón “crear puente”. Ejemplo: br0 y br1.

En el caso de crear el enlace *vLink*, debemos dirigirnos al menú “*vLink*” y seleccionar la opción “Agregar *vLink*”. Debemos introducir el nombre de los dos extremos y pulsar sobre el botón “crear vLink”. Ejemplo: vLink0 y vLink1.

Llegados a este punto, crearemos las cuatro interfaces de red para conectar nuestras máquinas virtuales. Para ello debemos dirigirnos hacia el menú “Interfaces” y seleccionar la función “Agregar interfaz”. Posteriormente, introduciremos los nombres de nuestras interfaces de red y pulsaremos sobre el botón “Crear interfaz”. Ejemplo: uml1, uml2, uml3 y uml4.

Por último, conectaremos todos los elementos que hemos creado: las interfaces de red y los *vLink* a los *bridges*. Para tal fin, nos dirigiremos al menú “Puerto” y pulsaremos sobre la opción “Agregar puerto”. Una vez introducidos los dispositivos, pulsaremos sobre el botón “Crear puerto”. Ejemplo: br0 con uml1 y br0 con vLink0.

Una vez realizados los pasos anteriores, tendremos lista nuestra topología de red de una forma sencilla y rápida. Además, podremos modificar las reglas de control de flujo y de esta forma filtrar los paquetes que circulan sobre ésta.

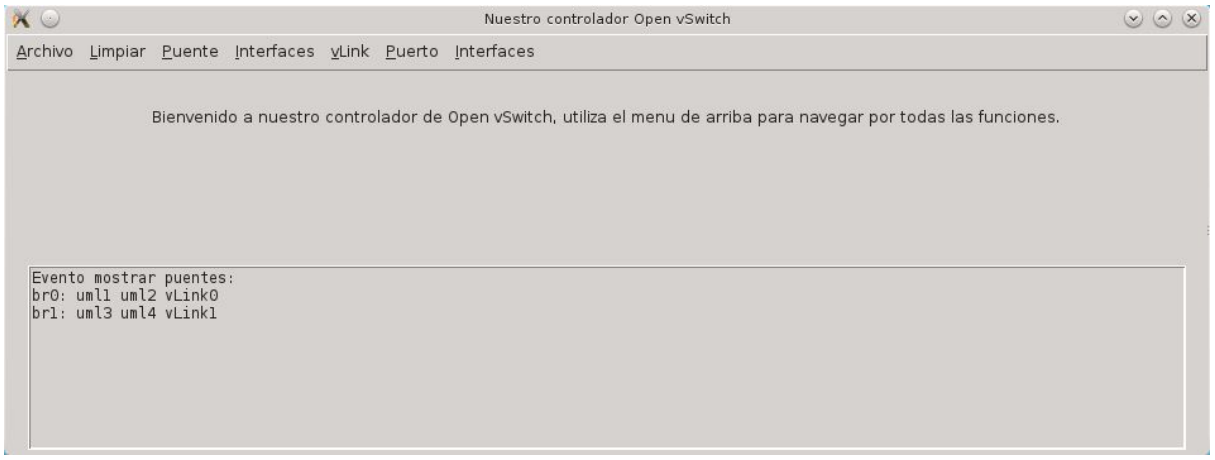


Figura 5.46 Resultado de caso de uso.

En la figura 5.46, mostramos la topología que hemos creado después de haber realizado todas las operaciones anteriores, mediante la función “Mostrar puentes” en el menú “Puente”.

# 6. Manual para un futuro desarrollo

En este apartado queremos hablar de las posibles mejoras que podría llegar a tener nuestra aplicación y que otros compañeros podrían seguir investigando.

La aplicación se conecta de manera local a Open vSwitch nuestra idea era poder mejorarla para que se conecte de forma remota mediante sockets.

El lenguaje básico es español, podría extraerse de cada una de las vistas en la capa de presentación el lenguaje en otro módulo aparte para poder integrar varios idiomas.

Muchas de las funciones piden *bridges*, interfaces o cables para poder eliminarlos, lo cual indica que la base de datos de Open vSwitch los conoce. Para agilizar el borrado, estas funciones podrían mostrar directamente en una lista los valores para poder seleccionarlos.

Crear una función que tenga como finalidad crear gráficamente una figura en la cual se represente los *bridges* virtuales, las máquinas conectadas a éstos y los cables que los conectan entre sí.

# 7. Aportación y conclusiones

Como aportación, hemos realizado un controlador de Openflow para usar de forma sencilla las órdenes de Open vSwitch, cumpliendo de esta forma con el objetivo de crear una aplicación para el ámbito docente, ya que nuestro controlador puede ser usado por alumnos de redes que quieren centrar su estudio en la creación de topologías de redes y el flujo de datos que pasa por ellas, sin necesidad de saber Open vSwitch de forma explícita, ya que éste requiere un estudio previo.

Realmente, nosotros pensamos que en este proyecto hemos recibido más conocimientos de los que hemos aportado, ya que la mayor parte del tiempo ha sido invertido en la investigación de Openflow y Open vSwitch. También lo aprendido de Tcl-Tk, ya que es un lenguaje que nunca antes habíamos utilizado y es en el que hemos basado y desarrollado nuestra aplicación, debido a que está disponible en un amplio rango de sistemas operativos, y es independiente de ellos, ya que, aunque cambiemos de versión sigue siendo compatible y funcional, lo cual facilita el mantenimiento y alarga su tiempo de vida.

Este ha sido el primer proyecto de gran envergadura, en el que hemos aplicado los conocimientos aprendidos durante la carrera. A continuación mostraremos todas las asignaturas en las que nos hemos basado:

- **Ingeniería del *software*, modelado del *software* y gestión de proyectos *software* y metodologías de desarrollo**. De éstas asignaturas hemos aplicado varios puntos:

    - La **metodología**: en nuestro caso nos hemos basado en metodologías ágiles para estar en contacto con el cliente de forma habitual y poder hacer cambios de requisitos sin provocar riesgos altos.
    - **Especificación de requisitos:** para analizar las necesidades de nuestro cliente y tener una base para poder empezar a desarrollar nuestro controlador.
    - **Plan de riesgos**: para descubrir cuales son los mayores riesgos que tenemos al crear nuestro controlador y evitar que lleguen a manifestarse.
    - **Planificación temporal**: para saber los tiempos que tenemos para realizar cada una de las partes de nuestra aplicación y evitar retrasos.
    - **Programación modularizada**: la estructura de cómo hacer un buen programa, dividiendo nuestra aplicación en tres partes (presentación, negocio y datos) para fomentar su mantenimiento.

- **Estructuras de datos y algoritmos**, **tecnología de la programación** y **técnicas algorítmicas en ingeniería del *software***: han sido totalmente necesarias para generar un código de forma correcta evitando, de esta forma, lentitud y bloqueos en la respuesta de nuestra aplicación.

- **Sistemas operativos**, **redes** y **administración de sistemas y redes**: ya que nuestro entorno de prueba era el sistema operativo Debian, era necesario saber cómo gestionarlo y tener conocimiento de cómo generar las distintas topologías de red, flujos de datos y manejar los distintos protocolos de red.

# 8. Bibliografía

## Documentación

[1] **Networking Bible** / Barrie Sosinsky / 978-0-470-43131-3
[2] **Apuntes proporcionados por el director** sobre Openflow y Open vSwitch.
[3] **VMware**, https://www.vmware.com/files/es/pdf/VMware-vSphere-Entreprise-Edition-Datasheet.pdf
[4] **Proxmox**, https://pve.proxmox.com/wiki/Open_vSwitch#Installation
[5] **Tcl/Tk programming for the absolute beginner** [Recurso electrónico] / Kurt Wall / 978-15-986-3438-9
[6] **Redes e Internet de alta velocidad**: rendimiento y calidad de servicio / William Stallings / 978-84-205-3921-8
[7] **Openstack**, https://www.openstack.org/software/
[8] **Neutron**, https://docs.openstack.org/mitaka/networking-guide/intro.html
[9] **Repositorio Github**: https://github.com/vicfd/TFG

## Imágenes

[10] **UCM**, https://www.ucm.es/data/cont/docs/3-2015-07-01-Marca%20UCM%20logo%20negro.png
[11] **Openstack**, https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/The_OpenStack_logo.svg/2000px-The_OpenStack_logo.svg.png
[12] **VMware**, https://www.muyseguridad.net/wp-content/uploads/2015/10/VMware.jpg
[13] **Proxmox**, https://www.proxmox.com/images/proxmox/Proxmox_logo_standard_hex_400px.png